Institutionen för datavetenskap
Department of Computer and Information Science

Final thesis

Improving software configuration management across multiple Microsoft Dynamics AX 2009 applications

by

Martin Cederbom

LIU-IDA/LITH-EX-A--15/020--SE

2015-06-10

Linköpings universitet

Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings universitet
581 83 Linköping

Linköping University
Department of Computer and Information Science

Final Thesis

Improving software configuration management across multiple Microsoft Dynamics AX 2009 applications

by

Martin Cederbom

LIU-IDA/LITH-EX-A--15/020--SE

2015-06-10

Supervisors: Petur Snaeland, Björn G. Karlsson
Examiner: Kristian Sandahl

Abstract

When working with software development in Microsoft Dynamics AX 2009 applications as an independent solution vendor (ISV) with more than a few products in the portfolio, the number of applications/installations to support the processes tends to be vast. Annata and To-Increase are both ISVs and face this situation; To-Increase for example has about 50 environments for one single version of Dynamics AX.

Change is inevitable in the software industry, regardless if the need originates from new features, bug fixes or change requests from customers; it requires modifications to existing products. Uncontrolled changes to the products must be avoided and any modifications should be introduced without affecting existing functionality.

This thesis investigates how two ISVs work with Dynamics AX and the existing tool Development Center and suggests improvements to the Software Configuration Management. The most beneficial change suggested is to create a single repository within the existing tool across the applications.

Acknowledgements

With no particular order in mind I would like to thank some of the people that made this thesis possible.

My supervisor Petur Snaeland for giving me the opportunity to do my thesis at To-Increase and for allowing me to take part in some genius work during the process. One of the greatest software architects I have had the pleasure of meeting.

Björn G. Karlsson for always supporting me in whatever I do, and for the many sessions together constantly increasing my knowledge. You are a true Master.

My examiner Kristian Sandahl at IDA, Linköpings University, and my opponent Alexander Örtenberg for reviewing, commenting and suggesting improvements.

Employees at both To-Increase and Annata for taking time for interviews and meetings with me.

Family and relatives for their support during the more intense periods, without you this would not have been possible.

My sister Charlotte Vainio for valuable thoughts on the thesis.

My children Oscar and Linnéa, you are the reasons I am breathing and why I wake up in the mornings (and during nights sometimes, come to think of it).

Through all this, supporting me and believing in me, my loving wife Linda that kindly left a blank space on the wall next to her framed master's diploma to motivate me to finally finish…

Table of contents

Chapter 1 Introduction 11
1.1 Background 11
1.1.1 Microsoft Dynamics AX 12
1.1.2 To-Increase 12
1.1.3 Annata 12
1.2 Purpose 13
1.3 Specified questions 13
1.4 Delimitations 13
1.5 Expected result 13
1.6 Intended readers 14
1.7 Method 14
1.7.1 The case study process 15
1.7.2 Theoretical method for data collection 16
1.7.3 Interviews 16
1.8 Critique of choice of method 17
1.9 Abbreviations and explanations 18
Chapter 2 Software Configuration Management 21
2.1 Configuration management history 21
2.2 Definition 22
2.3 Why use SCM? 23
2.4 SCM plan 24
2.4.1 Scope and purpose 24
2.4.2 Organization and resources 24
2.4.3 SCM activities 24

2.5 SCM Functionality Requirements 32
2.6 Anti-patterns and patterns for SCM 34
2.6.1 SCM Organization Pattern 34
2.6.2 Silver Bullet AntiPattern 34
2.6.3 Developer-Software CM AntiPattern 34
2.6.4 Decentralized CM AntiPattern 34
2.6.5 Requirements Jeopardy AntiPattern 35
2.6.6 Object-Oriented CM AntiPattern 35
Chapter 3 Result 37
3.1 Annata 37
3.2 To-Increase 38
3.3 Development Center 38
3.4 Microsoft Dynamics AX and version control 39
3.5 Stakeholder perspective 39
3.5.1 Current processes 39
3.5.2 Issues 42
3.5.3 Time consuming activities 45
3.5.4 Improvements 46
Chapter 4 Analysis 49
4.1 Current problem areas in relation to SCM activities 49
4.2 The SCM plan 50
4.3 Configuration items 51
4.4 Accounting and auditing 53
4.5 Controlling access 54
4.6 Process 54
4.7 Construction 54
4.8 How to tackle multiple applications 55
Chapter 5 Prototype for DevCenter Server/Client 57
5.1 Background 57

5.2 Software configuration management 57
5.2.1 Tracking needs 58
5.3 Assumptions 58
5.4 Technical solution 59
5.4.1 General idea 59
5.4.2 Web service 60
5.4.3 The web service class in AX – tpDevWebService 61
5.4.4 The Session – tpDevSession 64
5.4.5 Allow a form to use temporary tables 66
5.5 Further development 77
5.5.1 Clean up 77
5.5.2 Security 78
5.5.3 Sessions 78
5.5.4 Schema for update and refresh of data 78
5.5.5 Data model 79
5.5.6 Exclusive locks 79
5.5.7 Robustness 79
5.5.8 Elements 80
5.5.9 Configuration management 80
5.5.10 Persistent or non-persistent data 80
5.6 Setting the prototype up 81
5.6.1 Development Center AX2009 83
5.6.2 References 83
5.6.3 Business connector and Active Directory 83
Chapter 6 Conclusion 85
Chapter 7 After study 87
Chapter 8 References 91

Chapter 1
Introduction

This thesis discusses the need for Software Configuration Management and how it can be improved for companies working with software development in the enterprise resource planning system Microsoft Dynamics AX 2009, especially when there are multiple applications involved. The companies investigated, To-Increase and Annata, both have a vast setup of applications to be able to support their current product portfolios. The work with the thesis was initiated in the second half of 2008, a prototype implementing some of the suggested improvements was produced and recommendations were given in 2009. The after study was conducted in 2015.

In the sections below the background to the thesis is presented. The purpose of the thesis and specific questions are presented and the scope is set. Furthermore the chosen methods and references are discussed. At the end the outline is given to ease the reading and searching for specific information within the thesis.

1.1 Background

The use of computers and technology is playing a more and more important role for today's companies. Often the systems are complex, costly and critical for the business (Leon, 2000, pp. 47, 53). Many of the services provided by companies are partly or entirely dependent on computer systems. Can you even withdraw money from your bank account if the bank's computer system is down? Long gone is the time when the clerk wrote a receipt by hand and then handed over the money. Now you learn that the computer systems are down and that you are welcome back another day.

In the heart of the enterprises' computer systems is often the enterprise resource planning system (ERP-system). If the ERP-system fails it means unwanted and costly interrupts for the company.

Change is inevitable; Heraclitus claimed there is nothing permanent except change. Business or market conditions change, customer needs require new functionality and changes to the existing solutions, business organizations change, the budget and schedule change… (Pressman, 2005)

Being able to maintain large software products while reacting to ever changing surroundings and adding more value to customers without risking existing functionality is a must for most companies in the software business. Responding

quickly and efficiently to support issues, tracing defects and behaviors back to the original demands and modifications, avoiding interrupts and making sure all systems are up and running are some of the tasks at hand for the software development company, especially if they are in the ERP-business.

To-Increase and Annata are two software developing companies that build add-ons to the ERP-system Microsoft Dynamics AX. Both companies are Microsoft Gold Certified Partners and provide solutions that are certified for Microsoft Dynamics AX. Improving quality and the ability to provide desired customer functionality to the market as effectively as possible is a competitive advantage and a desired objective for most companies.

1.1.1 Microsoft Dynamics AX

Microsoft Dynamics AX is an ERP-system that has evolved from XAL/Concorde via Axapta to the product it is today. Earlier versions of Dynamics AX are designed for midsize and larger companies and offer a multi-currency and multi-language system. The core strengths are in manufacturing and e-business but the system also has great functionality for wholesale and service industries (Microsoft Corporation, 2008). Dynamics AX automates and streamlines financial, customer relations, and supply chain processes (Microsoft Corporation, 2002).

1.1.2 To-Increase

To-Increase, with headquarters in the Netherlands, is mainly focusing on developing solutions and components for Microsoft Dynamics products and Microsoft technologies. They provide products designed for specific vertical industry segments and sell their products via a partner network. Since 1999 they have been working with Microsoft Dynamics AX and today they employ approximately 50 persons. One of their products is called Development Center and is designed to assist organizations developing in Microsoft Dynamics AX. (To-Increase, 2008)

For the oldest Dynamics AX version there exist approximately 50 applications/installations in total, spanning 4 service packs.

1.1.3 Annata

Annata is an Icelandic company working with Microsoft Dynamics AX. They offer several vertical solutions, such as the Annata IDMS for Microsoft Dynamics AX. IDMS stands for Import and Dealer Management System and offers a vertical solution for companies involved with vehicles, heavy machinery, etc. The Annata

Group has about 60 employees in Iceland, Sweden, Denmark and the U.K. (Annata Sverige AB, 2008)

At Annata over 30 applications/installations in total are needed for handling the product portfolio – for one single version of Microsoft Dynamics AX.

1.2 Purpose

The purpose of the thesis is to suggest improvements to the software configuration management routines across multiple applications of Microsoft Dynamics AX 2009 for a small to medium sized software product developing company.

1.3 Specified questions

The resulting suggested improvements have been reached through the investigation of several questions. These questions are:

1. What challenges are introduced by the multiple applications setup from a software configuration management perspective?
2. What would be necessary to make software configuration management work across multiple Microsoft Dynamics AX 2009 applications?
3. How should a configuration item (CI) be defined in Microsoft Dynamics AX 2009?
4. What are the most beneficial improvements to be made to the current issue tracking?
5. What are the most beneficial improvements to be made to the current change management?

1.4 Delimitations

This thesis will be limited to focusing on the specific enterprise resource planning system Microsoft Dynamics AX. It will also focus on the management of changes to objects caused by new feature development, bug fixes, and customer specific requirements.

The basis for the suggested improvements shall be the existing tool Development Center.

1.5 Expected result

The expected outcome is a presentation of the software configuration management theories, a description of the current situation at the companies and an analysis with suggestions for improvements of the software configuration management routines in relation to Development Center. A prototype implementing the most important findings in Development Center should also be presented.

1.6 Intended readers

Readers that might find this content interesting and useful are people working with development in Microsoft Dynamics AX. Even though it is mainly intended for software product development organizations, part of the content should be applicable also to companies focusing on implementations at customers.

For people with a general interest in software configuration management and possible implementations, this thesis will provide information and examples for a specific case.

1.7 Method

To be able to describe the current situation at the companies I have chosen to conduct a case study. According to the Swedish dictionary Nationalencyklopedin, a case study is:

… a detailed study of a specific phenomena – e.g. a person or a group – in a research context and is used to give a balanced picture, enter deeply into and develop concepts and theories, sometimes also to illustrate and strengthen hypothesis. [1] (Nationalencyklopedin, 2008)

Runeson and Höst have defined guidelines for case studies in the field of software engineering and conclude it is a suitable research method since it "studies contemporary phenomena in its natural context" (Runeson & Höst, 2009).

In other research areas such as social sciences case studies have been well used and its methodology described in detail. Often these have been applied to the field of software engineering straight off. Runeson & Höst (2009) argues that there might be a difference in that way the study objects are more often:

1. Private corporations that are *developing* software rather than *using* software.
2. Project oriented rather than line or function oriented.
3. The advanced engineering work that is being conducted by highly educated people rather than simple routine work. (Runeson & Höst, 2009)

The matters investigated in this thesis all fit the descriptions from Runeson & Höst.

It is always a trade-of between the amount of control and the degree of realism when studying real life phenomena. Too little control in the study may lead to a result that cannot be interpreted properly, thus failing at understanding the studied object. Increased control reduces the degree of realism but can on the other hand lead to a scope that is too narrow, leaving important factors uninvestigated. Case studies are

[1] Author's own translation from Swedish.

targeting real life phenomena and may therefore be less controlled, but will have a high degree of realism. (Runeson & Höst, 2009)

There are different means of collecting data in relation to case studies, e.g. surveys, interviews, literature search, etc. For the case study itself, interviews and observations are mostly used. (Runeson & Höst, 2009)

Data collected for studies can be either *qualitative* or *quantitative*. The quantitative data can be analyzed using statistical methods, whereas the qualitative data needs sorting and categorization. Qualitative data collection tends to provide deeper and richer descriptions, and most case studies are based on that. (Runeson & Höst, 2009)

The purpose of the case study was to gain insight into and knowledge about To-Increase and Annata in general, and their processes when developing software in Microsoft Dynamics AX in particular. According to Avdic a case study is appropriate when an investigation doesn't have a narrow focus and a broader obtaining of information is desired (Avdic, 2005). The methods for gathering information have been observations and interviews as well as studies of various documents, e.g. internal guidelines and presentations. Information has also been collected during informal meetings and discussions and via instant messenger chats/calls.

1.7.1 The case study process

The process of conducting a case is defined in five major steps by Runeson & Höst (2009):

1. Case study design
2. Preparation for data collection
3. Collecting of evidence
4. Analysis of collected data
5. Reporting

In this report the objectives for the thesis itself was the base of the case study. Data collection was prepared by selecting an appropriate method, constructing adequate questions for finding descriptions about the objectives, gathering subjects for the planned interviews, and listing certain written material from the companies involved. The data collection was executed by performing the interviews and requesting written material. The findings can be found in Chapter 3 Result. Notes from the interviews as well as data in the written material have been analyzed and is the foundation of Chapter 4 Analysis.

1.7.2 Theoretical method for data collection

With the qualitative data collection method, it is important to use triangulation. Triangulation is about viewing the subject from more than one angle and can be used for data validation. Runeson & Höst (2009) mentions e.g. data source triangulation, where more than one data source is used for collecting data.

The data needed for the thesis have been of different character. A significant part has been concerning the understanding of To-Increase and Annata as organizations and their activities related to software configuration management, especially surrounding the tool Development Center. Interviews and document studies have been varied with informal meetings and discussions. We have met different stake holders and developers over a long period of time and discussed how the development progress from A to Z can be improved. Since we have been working with the development in Dynamics AX for a several years we have gained a quite profound knowledge in a short period of time. We are also a part of the business and we have decided to have broad and open questions in favor of specified questionnaires. This is to try to avoid biasing the result with the author's opinions. Still it might be a risk worth mentioning to readers, see 1.8 Critique of choice of method.

1.7.3 Interviews

Through meetings with the stakeholders from both companies, key personnel responsible for the software development routines where. The director of development/chief software architect and a senior solution architect/developer were interviewed from Annata. From To-Increase, the software architect/product manager and two senior developers where interviewed.

The questions prepared were as follows:

1. What are your main areas of work?
2. Can you describe what a normal day at work looks like for you?
3. Can you describe how :
    - A new feature is being handled from idea and approval to implementation and release.
    - A change request from a customer is handled from registration to closing the request?
    - A bug is handled from the initial discovery until it is solved.
4. What issues are you facing with the current setup of Dynamics AX?
    - From the infrastructure perspective? (Where are servers located, how are they accessible, how many environments do you have, etc?)

    - From the developer perspective? (Where are tasks stored, how are they accessed, what layers are you working with, how many environments, etc?)
5. What issues are you facing with the way you work today?
6. What are the three most time consuming parts of maintaining your products today?
7. If you could change three things, what would they be and why?
8. Do you have a formal description of your workflow?
    - Who is responsible for that workflow?
    - Where can I get a copy of it for further investigation?
9. Do you have policies for your respective environments?
    - Who is responsible for the policies?
    - Can you describe the policy?
    - Where can I get a copy of it for further investigation?
10. How many products do you maintain actively in terms of development?
11. How many versions of a product are maintained at the same time? (Versions: AX3-5, releases, customer installations, etc.)
12. How are these products distributed over different environments?
13. Is there anything you would like to add that I have not asked about?

From the interviews we received additional written information. All the interview subjects were informed about the purpose and intended use of the interviews. The interviews were recorded to facilitate follow up, which the subjects approved prior to the start. The recordings were later reviewed and key data from each question was identified, summarized and grouped under certain headings. It is presented in Chapter 3 Result.

1.8 Critique of choice of method

The author has been working with Microsoft Dynamics AX since 2004; first as a customer and later as project manager, responsible for technical aspects and developer. Due to this the result might be biased by the author's opinions with both pros and cons. Interviews and discussions and the result thereof might benefit from the author's prior knowledge of the subject as they can go beyond the surface and more deeply penetrate the essence of the matters. On the other hand, certain old standpoints and methods might not be challenged to the same extent as if someone outside of the subject area would have investigated the topics at hand.

Annata has a Swedish subsidiary focused on consultancy services and customer implementations where the author is currently employed. The business is separated

and the Icelandic headquarter is the product development company investigated in the thesis.

Case studies are often criticized for being of less value than analytical and controlled empirical studies. By applying proper research methodologies and keeping in mind knowledge is more than statistical significance some of that can be remedied. (Runeson & Höst, 2009)

1.9 Abbreviations and explanations

In this thesis certain abbreviations and words will be frequently used. Here is a table with the definition we use.

Abbreviation	Description
AOD	Application Object Data. A file with extension .aod, meaning Application Object Data. It contains Dynamics AX application data; the model element, source code, etc. Used to deliver solutions.
AOS	Application Object Server – server used by Dynamics AX
AOT	Application Object Tree – a tree-view of all elements in Dynamics AX.
APL	Application platform, product from Annata
APP	Application productivity platform, product from to-Increase
BPL	Business platform, product from Annata
BUS	Layer in Dynamics AX
CM	Configuration Management or Change Management (see also SCM)
CNT	Connectivity Studio, product from To-Increase
Development Center	Development StudioCenter, or DevCenter, software developed by To-Increase. The earlier version was called Development Toolkit. The new version was later renamed to Development Studio, DevStudio.
Dynamics AX	Microsoft Dynamics AX, enterprise resource planning system from Microsoft.
Element	The unit in which metadata and source code are stored in the .aod file.
IDMS	Import Dealer Management System, product from Annata
IEM	Industrial Equipment Manufacturing, product from To-Increase
IIS	Internet Information Services, web server from Microsoft
ISV	Independent Software Vendor
Layer	DAX Dynamics AX has several layers in which application object

	data is kept. Partners and customers may access different layers. Layers are ordered from the system layers via BUS, VAR and CUS to USR. See AOD.
RCM	Retail Chain Manement, product from To-Increase
Release	Official build of a solution or product/component
SCM	Software Configuration Management (see also CM)
Solution	A larger software product. May be a bundle of smaller products or components.
USR	Layer in Dynamics AX
VAR	Layer in Dynamics AX
XPO	A formatted text file containing mainly elements. Has the extension .xpo and is an export format from Dynamics AX.

Chapter 2
Software Configuration Management

What is software configuration management? Pressman says change management is more commonly called software configuration management and uses the acronym SCM or CM for it (Pressman, 2005). That description implies that they are the same thing. There seem to be considerable overlap and confusion between change management, change control and configuration management. Gartner calls SCM software configuration management, but notes it is also known as software change management. They define it as "SCM is a methodology for software problem/change request initiation and tracking; change impact analysis; version control; security administration of software assets; software promotion; quality reviews; and software distribution" (Gartner Inc, 2015). The terminology might be a bit confusing, but in this thesis I use the term software configuration management and further down in this chapter I define what I mean with the term.

The concept of software configuration management is originally sprung out of configuration management. What is configuration management (CM) then and how has it evolved?

2.1 Configuration management history

Configuration management was originally used as techniques and disciplines in the defense industry environment for resolving issues with poor quality, incorrect parts being ordered and parts not fitting, all problems that in the end were causing costly budget overruns. This was in the late 1950s and early 1960s and was a response to the need of identifying and controlling the design of increasingly complex equipment, and a way to communicate the information between involved engineers. Earlier the systems were simpler and one could rely on the discipline of individuals. Now the work could be initiated by one engineer and then carried out by another, and the projects could span several years. (Berlack, 1992)

Since the middle of 20th century there have been various standards developed and refined in the area of configuration management. Commercial standards are now available; e.g. from the Electronic Industries Alliance (EIA), the Institute of Electrical and Electronics Engineers (IEEE), and the International Organization for Standardization (ISO) (Moore, 1998). In the beginning the standards mainly focused

on requirements and needs for hardware. As the software became an integrated component of the hardware, the need for management of software evolved.

2.2 Definition

According to Berlack the primary activities of configuration management are *identification*, *change control*, *status accounting*, and *configuration audit*. He also argues that when interface and subcontractors are part of a project, *interface control* and *subcontractor CM control* should be added to the activities. (Berlack, 1992)

Pressman describes the SCM as an umbrella activity that is applied during the entire software process. He defines it as activities developed to *identify change*, *control change*, ensure that change is being *properly implemented*, and *report changes* to others who may have an interest. (Pressman, 2005)

Even though expressed differently, they identify the same activities. Since I have chosen to use the term "software configuration management", let us look at the words involved.

A formal definition of the term *configuration* can be found in MIL-STD-480B (Department of Defense - United States of America, 1988):

> "The functional and physical characteristic of hardware, firmware, software or a combination thereof as set forth in technical documentation and achieved in a product."

The term *configuration item* is then defined in the same military standard as:

> "An aggregation of hardware, firmware, software, or any of its discrete portions, which satisfies an end use function and is designated for configuration management. Configuration items may vary widely in complexity, size and type […] Configuration items are those items whose performance parameters and physical characteristics must be separately defined (specified) and controlled to provide management insight needed to achieve the overall end use function and performance."

Berlack summaries this and says that a configuration item is a stand-alone, test-alone, use-alone element (Berlack, 1992).

MIL-STD-973 (Department of Defense - United States of America, 1992) states that that *configuration management*, from the configuration item perspective, is a "discipline applying technical and administrative direction and surveillance over the life cycle of items to

1. Identify and document the functional and physical characteristics of configuration items.

2. Control changes to configuration items and their related documentation.
3. Record and report information needed to manage configuration items effectively, including the status of proposed changes and implementation status of approved changes.
4. Audit configuration items to verify conformance to specifications, drawings, interface control documents, and other requirements."

Configuration management is not to be seen as means solely for software product control, it is a technique to control the overall management process. (Moore, 1998)

2.3 Why use SCM?

What motivates the use of SCM in a project? Quality demands alone might motivate the use of SCM, but there are more reasons of course. Without going too much into the software engineering principles with the classical waterfall model versus more iterative approaches, the trend today in many companies in the software development industry is towards a more agile and iterative process. Royce claims that change management is critical to iterative processes; tracking changes to configuration items is crucial to be able to track the progress of creating an acceptable end product. He continues and states that "change management has become fundamental to all phases and almost all activities" (Royce, 1998).

Changes will always occur in software development, and most of them are justified (Pressman, 2005). The origin of the changes can be anything basically, but Pressman identifies four fundamental sources: new business or market conditions, customer needs, reorganization or business growth/downsizing, and budgetary or scheduling constraints (Pressman, 2005). Product development companies as well as companies involved with customer implementations all face ever changing environments. Managing changes is important and Pressman claims that a well-run software project easily can turn into chaos by a stream of uncontrolled changes (Pressman, 2005). Pressman states that a primary goal of software engineering is to make it easier to implement changes with less effort. Change management is the set of actions that will make it possible for engineers to reach the goal. (Pressman, 2005)

According to Royce one of the key practices to improve overall software quality is providing integrated life-cycle environments that support early and continuous configuration control and change management. He also mentions support for rigorous design methods, document automation, and regression test automation as key factors. (Royce, 1998)

2.4 SCM plan

"Failing to plan is planning to fail" as the saying goes. An SCM plan is needed to document the procedures, duties and responsibilities decided upon (Leon, 2000). The SCM plan describes how configuration management is to be performed and there are several standards available. Berlack (1992) describes the basic outline of such a plan:

- Scope and purpose
- Organization and resources
- CM activities for
    - Identification
    - Control
    - Status accounting
    - Configuration audits
- Subcontractor control
- CM milestones/schedules
- Notes and appendixes

The sections in this chapter follow the same outline.

2.4.1 Scope and purpose

This section provides a background and reasons as to why SCM is to be used for the current project. It describes the project with its objectives and purpose and gives an outline of the plan. (Berlack, 1992)

2.4.2 Organization and resources

The organization chart of the project is presented and this section describes at what levels configuration management are to take place in the organization. The resources section describes the persons involved and includes the authority and responsibilities of the different positions in the SCM organization. Under this heading one also describe any interfacing organizations, such as a support or software quality assurance organization. (Berlack, 1992)

2.4.3 SCM activities

Below are the activity sections as outlined by Berlack (1992).

2.4.3.1 Identification

The identification activity is concerned with identifying the configuration items and their components. Capturing specifications and documents related to the software being developed and putting them under configuration management is described. Requirements traceability is important and the configuration items interrelationships

must be documented in some way. Both top down and bottom up traces need to be available on request, and Berlack (1992) suggests a parent-child relationship approach but notices that the system gets more complicated when interfacing elements and changes are introduced.

To be able to control and manage change to configuration items, they have to be identified and organized in some way (Berlack, 1992), (Pressman, 2005). Pressman suggests a text string identifying each configuration item and classifies the items as being basic or aggregate items to organize them in an object-oriented fashion. A basic item, or object, can be any unit of information created during analysis, design, code, or test. It could be source code for a component, part of a section of a requirements specification or a suite of test cases. A grouping of basic objects and/or other aggregate objects constitutes an aggregate object. (Pressman, 2005)

Berlack stresses the importance of carefully selecting the configuration items. The selection and creation of the software hierarchy should be done early in the process and may impact the amount of work needed for maintenance later in the lifecycle. Grouping functions into several dissimilar configuration items can lead to extra work when changes are introduced and the opposite is true; proper selection can limit the efforts needed since the functionality is isolated to software units within fewer items. (Berlack, 1992)

2.4.3.2 Change control

In this section it is declared when control is supposed to be initiated on a document, source code etc., and the item thereby be put under configuration control and becoming a configuration item. This is an important activity. The process of performing this activity should be described and documented. Berlack presents a suggested software change flow (Berlack, 1992):

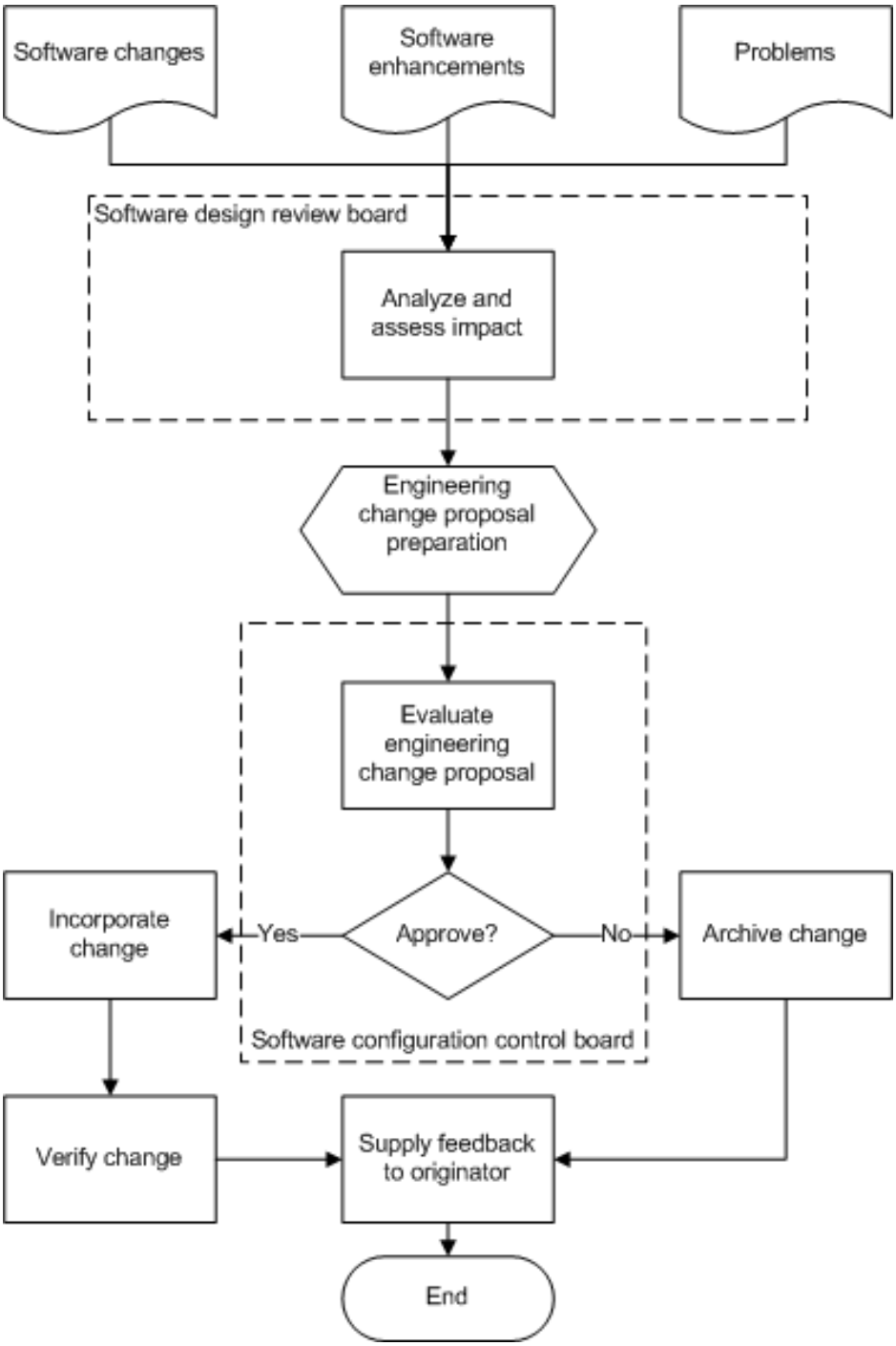


Figure 1 Change flow as suggested by Berlack (Berlack, 1992)

This section should also describe the change configuration board, its named members and their respective functions and authority.

It is vital to control change: a chain of uncontrolled changes can easily lead to chaos in a software project. However, too rigorous control processes can lead to decreased creativity and impede progress. (Pressman, 2005)

As seen in *Figure 1 Change flow as suggested by Berlack*, a general change control activity can be simplified into a flow chart. Pressman has a similar flow describing

the change control process. He also refers to the version control process, which he sees as a separate process step even though interrelated as a subordinate activity under change control. Version control is about tools and procedures to manage different versions of configuration items. A version control system should consist of four major features (Pressman, 2005):

1. A project database or repository to store configuration items.
2. Version management capability to handle version history.
3. A make facility to enable the construction of specific versions from relevant configuration items.
4. Issue or bug tracking capabilities.

The version control system comes into play when a change has been approved and objects are about to be modified. By using a simple check in/out approach for configuration items the version control mechanisms can provide access control and synchronization control. (Pressman, 2005)

Berlack treats internal changes as a separate process, less structured and with no suggested standard. He describes a workflow which basically follows the same pattern as a more formal change process and stresses that it is necessary to record the changes regardless. (Berlack, 1992)

Pressman refers to informal change control, something to apply before a configuration item is treated as a baseline. Many people fear the bureaucracy introduced with software configuration management, and it is important to find the appropriate level. For informal change control the developer may decide on minor changes himself as long as it doesn't affect broader system requirements. After a formal technical review and approval of the changes made, a baseline can be created. Thereafter project level change control is implemented. In order to make a change, the project manager or change configuration board must be consulted for approval. Once the software has been released to customers, formal change control should be applied. (Pressman, 2005)

2.4.3.2.1 Change control for web applications

Pressman describes web engineering as somewhat different to traditional software engineering. It uses iterative and incremental approaches and applies many principles from the agile software development discipline. Changes in the agile web engineering world are viewed differently than in a conventional software project. The change control and configuration management in general can, as already mentioned, appear to be too bureaucratic and formal for the agile process. Pressman states that configuration management principles, practices and tools must not be rejected, but rather molded to meet the special needs. (Pressman, 2005)

How do you manage the many iterations and continuous stream of changes then? Pressman suggests a categorization of each change into one of four classes (Pressman, 2005):

- "Class 1 – a content or function change that corrects an error or enhances local content of functionality
- Class 2 – a content or function change that has impact on other content objects or functional components.
- Class 3 – a content or function change that has broad impact across a WebApp (e.g. major extension of functionality, significant enhancement or reduction in content, major required changes in navigation).
- Class 4 – a major design change (e.g. a change in interface design or navigation approach) that will be immediately noticeable to one or more categories of users."

Once categorized, the workflow described below can be used to handle the different classes.

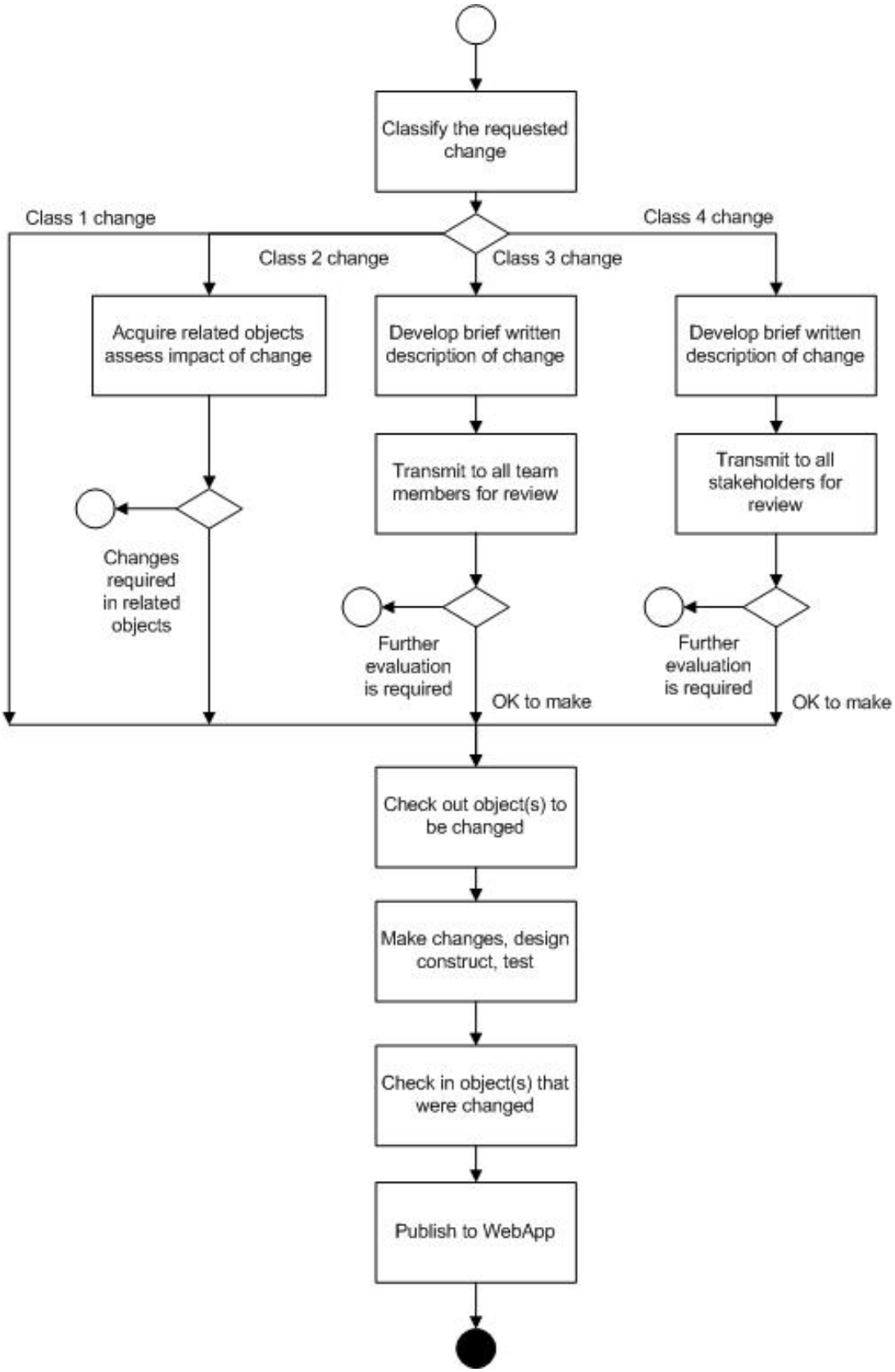

Classify the requested change
Class 1 change
Class 2 change
Class 3 change
Class 4 change
Acquire related objects assess impact of change
Develop brief written description of change
Develop brief written description of change
Transmit to all team members for review
Transmit to all stakeholders for review
Changes required in related objects
Further evaluation is required
OK to make
Further evaluation is required
OK to make
Check out object(s) to be changed
Make changes, design construct, test
Check in object(s) that were changed
Publish to WebApp

Figure 2 Managing changes for WebApps (Pressman, 2005).

Pressman also talks about version control. A web application may exist in several versions at the same time: one version is live and already in use by end-users,

another version is in the final stages of testing and may contain the latest increment from development, and yet another version may be under development in the development environment. Pressman stresses the importance of identifying each of the configurations item with its appropriate version. Both version control and change control must be in place. (Pressman, 2005)

2.4.3.3 Status accounting

Status accounting is important for knowing what changes have been applied to a certain item and to know the latest revision of any item. Being able to query this data and retrieve information on how many items have been worked on, their respective status, if they have been approved or rejected or if work is still to be done, are all useful information to provide service to the project.

Status accounting, or configuration status reporting as Pressman refers to, is an activity intended to answer the questions: what happened, who did it when and what else will be affected? Associated to this is a flow of information. For each identification change (assignments or other modifications) to a configuration item, a configuration status reporting entry is made. Each time a change is approved by the change configuration board an entry is made as well as when configuration audits are conducted. (Pressman, 2005)

Berlack describes status accounting as the recording activity and it "follows up on the results of the configuration management activities". This activity helps the user to keep track of the current configuration of delivered software, status of suggested changes, whether they are approved or reviewed or the implementation progress for a certain change. It is important to be able to track the status both bottom up and top down according to Berlack. (Berlack, 1992)

Status accounting is information and can assist project management in decision-making. It can be used to tell what major problems there are, how the project is progressing, what types of changes and their respective causes, how many of these have been approved, and at what rate changes are introduced. (Berlack, 1992)

Status accounting can be a major benefit for maintenance purposes. Berlack reasons that maintenance starts as soon as the first change has to be made, and may very well be during early testing phases. To introduce new changes, the history of what has been done previously must be known to make sure pitfalls are avoided and that the changes suggested still conform to requirements. An equally important motivation for status accounting is keeping project members up to date with information about what is being developed, what is ready for test, etc. Berlack also mentions assured

supportability of released products as one the benefits with status accounting. (Berlack, 1992)

2.4.3.4 Configuration audit

The SCM plan should describe how the outcome of the development process will be tested and examined, and how that testing is to be performed to ensure that outcome conforms to the requirements. It should also describe how the documentation associated to the product is reviewed to make sure it describes the actual work product being delivered. The plan should describe how this audit should be conducted, by whom and with what authority.

Configuration audits are part of the answer to the question: how can we ensure that a change has been implemented properly? The other answer is formal technical reviews, according to Pressman. There is a gap between the control mechanisms being applied up until a change has been approved and the control during the actual implementation of the change. The software configuration audit complements the formal technical review and helps bridging the gap, ensuring that what has been implemented is also what actually was intended. (Pressman, 2005)

Berlack divides the audits into two types, the functional and the physical configuration audit. The functional configuration audit's objective is to verify that the functionality of a certain software configuration item conforms to its requirements. This is done by reviewing data from test reports and other documents, and comparing it to statements in documents such as the requirement specification. It's also important to check interrelationships, for example to verify that user documentation for a certain requirement contains the latest changes. (Berlack, 1992)

The physical configuration audit focuses on determining if design and product specifications, as well as other documents, represent the actual software created. (Berlack, 1992)

2.4.3.5 Subcontractor control

This section is involved in describing how configuration management control is to be applied to any subcontractor and how the internal demands should be mirrored to the subcontractors.

2.5 SCM Functionality Requirements

Susan Dart (Dart, 2007) describes functionality requirements for a configuration management system. In figure 3 below, each box represents one major area of functionality as identified by Dart. All these requirements are seldom or never to be found in one single system, but make up the requirements of an ideal system. The team and process boxes are the significant areas since they affect or are affected by the other functionality areas according to Dart. (Dart, 2007)

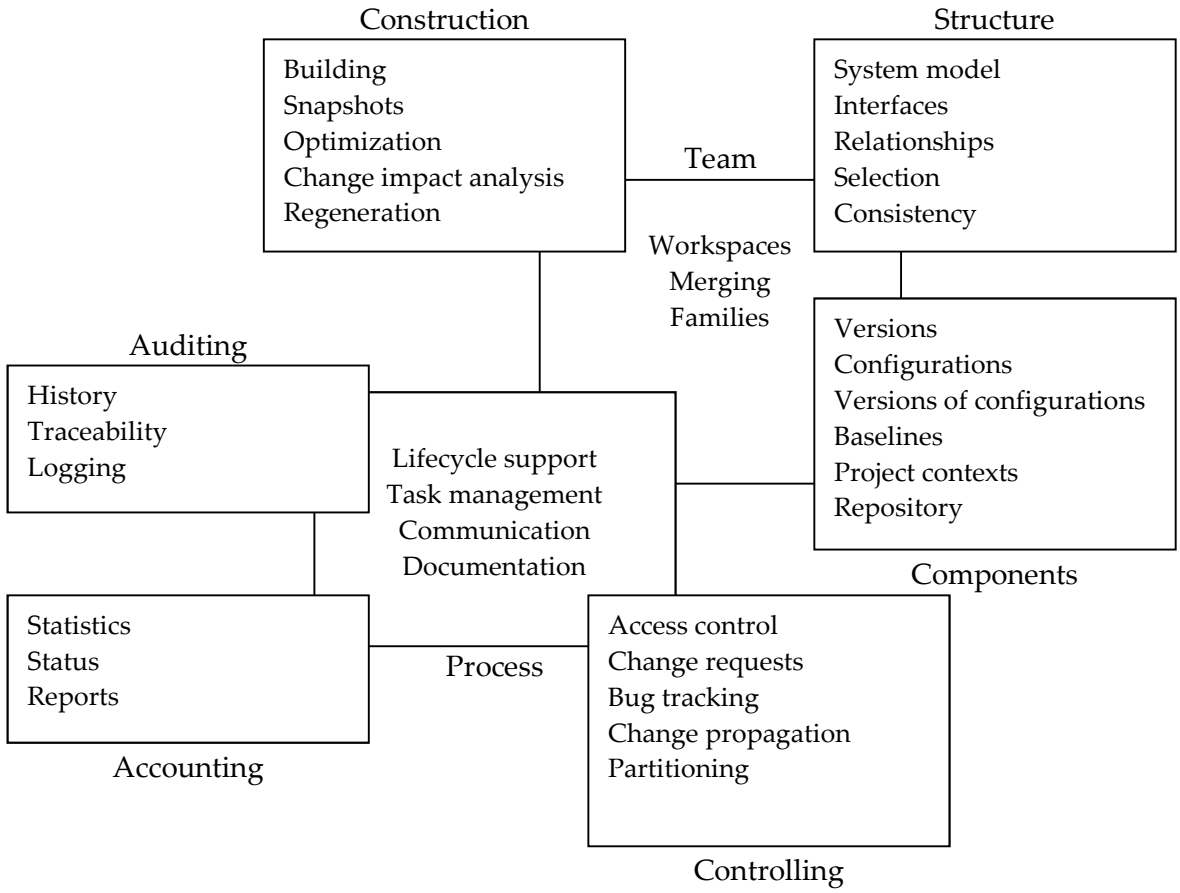


Figure 3 Describing CM Functionality Requirements, (Dart, 2007)

The major areas of functionality are described below (Dart, 2007):

- **Components**: identifies, classifies, and accesses the components that make up the software product.
- **Structure**: represents the architecture of the product.
- **Construction**: supports the construction of the product and its artifacts.
- **Auditing**: keeps an audit trail of the product and its process.
- **Accounting**: gathers statistics about the product and the process.
- **Controlling**: controls how and when changes are made.
- **Process**: supports the management of how the product evolves.
- **Team**: enables a project team to develop and maintain a family of products.

The component area of functionality holds requirements such as recording versions of components, their differences and the reasons for the differences. It also includes the identification of groups of components, groups that make up a configuration. Versions of these components are also identified. In this area one will also find baselines for a product, and the project context to which collections of components and artifacts are related. The components need some kind of repository or library once under configuration management control. (Dart, 2007)

Structure is the functionality area where users can model the structure of a product via a system model. In the model it should be possible to specify interfaces among the components, versions, and configurations. By doing this, the components become reusable. Support for identifying and maintaining relationships between components is required as well as the possibility to create sound versions of a product by selecting compatible components. (Dart, 2007)

Construction is concerned with providing means to build and construct the product. It should be possible to freeze the development at a certain stage, e.g by creating snapshots. Optimization is about providing mechanisms to assist in way the product is built, to minimize the need of recompilation of components. This area of functionality should help the user analyze the impact of a proposed change before any modifications are realized and support possibility to go back to a certain stage in the lifecycle. (Dart, 2007)

For auditing all changes must be stored as historical data, a trail should be kept so tracing is made possible between related components in the product and their evolution. It is also important to store a detailed log of the work done. (Dart, 2007)

Accounting is about recording statistics for the product, providing tools to allow the user to examine the status of, and generate reports for, different aspects of the product. (Dart, 2007)

The functional area of controlling should provide the user with tools to control access to components to avoid unmanaged changes or conflicts in changes. The toolset should also provide for on-line change requests and bug tracking with the ability to trace what is assigned to whom, and who is working with what, when and where. Changes should only be allowed to propagate over different versions of products in a controlled way and means to support this should be included in the toolset. It is also desirable to limit the effects of changes to the product, and this can be done by partitioning. (Dart, 2007)

The significant process functionality area should provide the user with support for the organizations policies and lifecycle models. Means for assisting in identification and tracking of tasks and their statuses is needed and the tool should facilitate the

communication of information about relevant events to the appropriate users. It should also provide facilities for documenting knowledge about the product. (Dart, 2007)

The other more important area is the team area. Users need personal and group workspaces, tools for merging multiple changes and to perform conflict analysis. Also, support for grouping products into families of product is desirable. (Dart, 2007)

2.6 Anti-patterns and patterns for SCM

In the area of SCM, patterns and anti-patterns ("do's and don'ts") have been documented. Below a few of these are briefly mentioned, since I believe these simple statements can be of help.

2.6.1 SCM Organization Pattern

Define the SCM to establish organizational values – SCM goals, organization structure, processes, standards, authority and responsibility. (Brown, 1999)

2.6.2 Silver Bullet AntiPattern

Addresses the mistaken belief that a tool can solve all of your configuration management problems. (Brown, 1999)

2.6.3 Developer-Software CM AntiPattern

Identifies what occurs when CM is left to the development team, rather than to an independent CM organization as it should be. The anti-pattern teaches us to make sure SCM is not only something done for the code itself. The product is more than source code; documentation of tests, designs, user manuals, etc are also artifacts that constitute the product. (Brown, 1999)

2.6.4 Decentralized CM AntiPattern

Provides insight into the value of a central repository for performing development and configuration management. The benefit with the central repository is the ability to share information and to make it available to all users. (Brown, 1999)

The anti-pattern stresses the importance of a central repository. The solution to the anti-pattern is to gather everything in one place; make information available in one single tool. Avoid having bits and pieces spread over the entire infrastructure. (Brown, 1999)

2.6.5 Requirements Jeopardy AntiPattern

From this anti-pattern we learn that we should make sure to submit the requirement baseline to CM. (Brown, 1999)

2.6.6 Object-Oriented CM AntiPattern

This anti-pattern tells us to keep track of both the component level and the code level. This is even more crucial for object oriented development. Different levels of granularity in the software configuration management are needed to reflect the object oriented reality. (Brown, 1999)

Chapter 3
Result

To gain knowledge about Annata and To-Increase I have conducted interviews and gone through internal documents. The interviews took place in September 2008. Information has also been collected during informal meetings and discussions and via instant messenger chats/calls.

3.1 Annata

At Annata they are involved in product development as well as customer implementation projects and consultancy activities. For their product development Annata uses the To-Increase product Development Center to organize and handle tasks for each product. On top they have an issue management system which is publically available to customers via a web front end. Maintaining the connection between the two systems is manual at the moment and the process is described in their internal guidelines.

There are numerous solutions identified as products and currently maintained at Annata. Each product is maintained within a separate development environment where the Development Center is installed. For each product development environment exists a release environment. The purpose of the release environment is to allow only certain approved modifications to be released to customers and partners, since the development environment is constantly evolving.

The products are also componentized in the sense that one standalone product can be a component of another product.

Over 30 installations in total are needed for handling the product portfolio, for one single Dynamics AX version. The current maximum number of AOSes on a physical server is 30, forcing Annata to use several different machines to handle their environments. During the beginning of 2009 their products spanned 3 versions of Dynamics AX: AX3, AX4 and AX2009.

For the customer projects there have been experiments with using Development Center and its predecessor Development Toolkit (see 3.3 Development Center). Even Rapid Configuration Tool, nowadays a Microsoft product, has been evaluated. Normally, however, there is no general Dynamics AX system support on site for the

change management process. The gap-fit lists are maintained in Excel spreadsheets; documentation and change requests are created and scattered around file shares and personal folders.

3.2 To-Increase

The focus of To-Increase is product development. Their largest solutions for the Dynamics AX area are called Industrial Equipment Manufacturing, IEM, and Retail Chain Management, RCM. They have a far evolved component strategy for their development; the main products are based on smaller and standalone components. These components in turn are shipped and treated as individual products. Both IEM and RCM are certified for Microsoft Dynamics AX and are Microsoft Dynamics Industry Solutions, meaning they have passed quality tests and conform to the best practices provided by Microsoft.

The current strategy is to separate development efforts for the different components.

In total To-Increase have about 7 Dynamics AX 2009 environments; 4 development, 2 test and 1 release environment. The AX3 version range 4 service packs and sums up to approximately 50 environments. They use separate environments for testing purposes and these are included in the total count.

To-Increase has developed a product named Development Center and also Development Toolkit (see 3.3 Development Center).

3.3 Development Center

Development Center (DevCenter) is a product developed by To-Increase to support the development process in Dynamics AX. It is available for Dynamics AX 4 and Dynamics AX 2009. The predecessor was called Developer Toolkit (DevToolkit) and was available on Dynamics AX 3.

DevCenter is built on other components in the To-Increase Suite, making it very flexible but also dependent. The two components are the Productivity Platform and the Workflow Management. Productivity Platform is a powerful set of consistent, reliable, reusable building blocks for Dynamics AX components or solutions. It acts like a fundamental layer that separates a solution from core Dynamics AX.

Workflows can assist in automating the development processes. The Workflow Management supports sequential workflows using events, actions and action flows, and also state-machine workflows, using lifecycles and events on state transitions. The component can route tasks, alerts and approvals to people and execute any kind of operation based on configuration. Therefore the DevCenter can be used with little or no need for customization. With these features DevCenter can be configured to

support almost any needs. And of course, DevCenter could be extended if needed, just as any Dynamics AX application.

DevCenter provides functionality for task management with workflow, version control of individual elements, automation of tasks, and it provides a set of tools to support the developer in the everyday work.

3.4 Microsoft Dynamics AX and version control

The current versions of the applications maintained at Annata and To-Increase span Dynamics AX 3, AX 4 and AX 2009.

Support for version control for source code within Dynamics AX is as follows:

- AX3 – non existing
- AX4 – requires MS Team Server and e.g. MS Visual Source Safe
- AX2009 – offers the same version control system as AX4 but also has built-in support with MorphX

3.5 Stakeholder perspective

3.5.1 Current processes

This section describes the current processes at the companies.

3.5.1.1 Annata

Annata uses a service system for handling issues. This service system is part of one of the solutions they offer to the market and it is also used internally to manage all kinds of issues.

The development process is well documented. They strive to be as agile as possible and their process can be described in the following schematic picture:

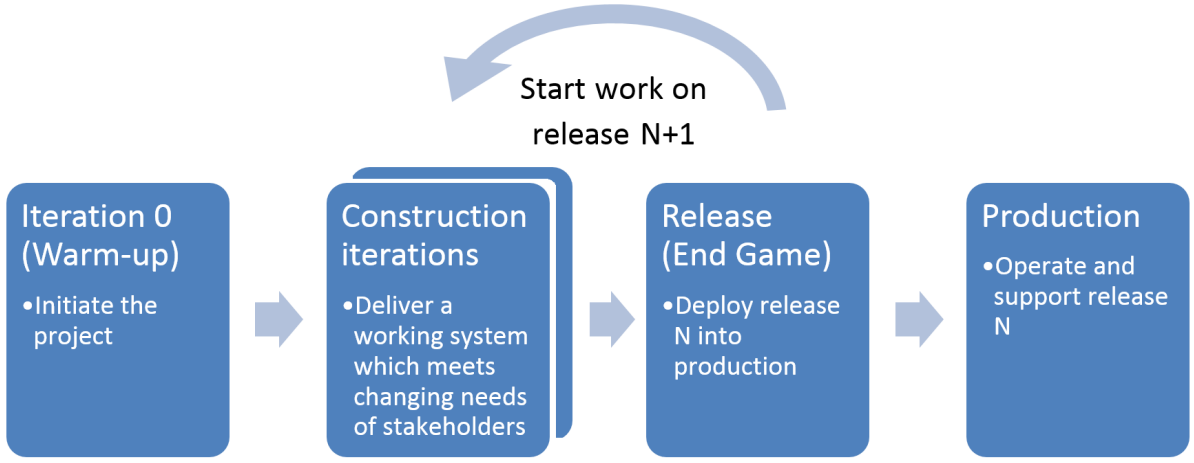

Start work on release N+1
Iteration 0 (Warm-up)
•Initiate the project
Construction iterations
•Deliver a working system which meets changing needs of stakeholders
Release (End Game)
•Deploy release N into production
Production
•Operate and support release N

Figure 4 Annata's development process

As figure 4 describes, the work is done in iterations. The development and design efforts are encapsulated in the construction iteration. Annata has an experienced team with in-depth knowledge of development, the application itself and real life experience of the areas involved in their products. With the teams organized as described, they can get a good throughput and quickly respond to new requirements. One of the keys is the active participation of the stakeholder. The document that describes the development process also describes how Annata work with Development Center.

Defect reports and requests for new features interact with the process described above and they have different processes, but in reality there is little difference in how they are treated.

All issues should start as a service request in the service system. The employees describe the issue at hand with the words of the customer/requester. The requester is associated with the service request and the request is classified. Some of these tasks might never become developer tasks as they are solved by other means. However, they set a developer as responsible for the issue. Since Annata has multiple environments with different components maintained in different environments, part of the initial analysis is deciding to what system the issue belongs. In the environment they identify, Development Center is installed. They create a development task in Development Center and link it manually to the originating service request. They describe if the fix has to be back ported to previous version of Dynamics AX and they try to add a note if the task solved would be beneficial for other customers as well. To gain the most from any modifications, Annata tries to generalize the problem at hand to be able to find a generic solution. If a generic solution can be found that affects other customers as well, it may be decided that the issue should be solved as part of the product development. If not, it may be decided

that the solution should be applied locally at the customer site and remain a customer specific customization.

Defects or bugs might be assigned higher priority and are more likely to result in a hotfix than other issues. When the associated task is solved and is linked to a service request, the service request it updated manually.

They try to be agile, so design and details might be specified on the fly. During this process and any modifications, they try to keep product documentation up to date at the same time. Recently Annata has started to conduct a list of new features they are adding to be able to describe functionality for new versions in a user friendly way.

3.5.1.2 To-Increase

The existing routines at To-Increase are said to be documented by the interview subjects. However, these documents are not easily available or accessible and even though developers are aware of them no one can say how to retrieve the latest version. It seems that many efforts have been made but a formal companywide approved document describing the routines does not exist. They do however have informal routines and processes that they follow. The processes differ a bit depending on the product. IEM and RCM, the major products, have more ceremony associated with the new feature, bug and change request handling than smaller components.

Requirements from a customer can be handled differently, depending on the need. There may be features needed to e.g. win a deal, such as prototypes, design, or a proof of concept. Some issues may be rush service requests (high priority issues) to solve emergencies at the customer site.

In general, ideas for new features come from various sources such as product managers, customers, support or developers. The process for handling the new feature suggestions varies between different products. Some products are certified by Microsoft and thus require stricter handling before changes are introduced. Normally all these features should be documented and then handled properly, but that is not always the case. At To-Increase they do have good internal understanding of the processes, but not everything is done by the book; it is not a completely documented process and the current handling is described as “consistently inconsistent” to quote one of the interview subjects.

To-Increase uses a website for support issues. The issues are validated and if they should be solved by means of code changes it should normally end up as a task associated to a component and version in Development Center. In the system,

specific workflows apply for the tasks and there are routines for documentation of hotfixes and new releases.

Change requests may be new features, depending on if it fits within the product or not. If a change request is deemed suitable as a new feature for the product at hand it is processed further, if not the change request is returned to the customer without a fix.

To-Increase used a web portal both for the support issues but also for other customer related communication, such as providing information about updates to their products, hotfixes available, etc.

Normally the need for a change is described by a consultant. The need is reviewed and discussed by e.g. solution architect, senior developer and product managers to identify if it can be solved in a generic way, and or whether it should be part of the product or not. A decision is taken regarding the feature in terms of e.g. validity against target markets. The design idea may be approved and if need be it will be reviewed from a technical viewpoint before a detailed design is created.

To document and record new changes, different techniques are used such as spreadsheets and DevCenter.

3.5.2 Issues

3.5.2.1 Annata

3.5.2.1.1 Infrastructure

At Annata they have decided to deliver solutions by layer and to supply hotfixes as xpo-files. Many of Annata's solutions consist of different standalone products mixed together. To be able to deliver layers, they need at least one environment per solution to function as a layer factory. For each development environment they need one AOS-instance. To make sure development can continue while stabilizing a release, Annata has decided to use release environments for their solutions, and these release environments fill the purpose of a layer factory. The release environments each require their own AOS. In total they maintain about 10-15 release environments, and with a development environment for each it adds up to 20-30 environments. This is only for one version and service pack of Dynamics AX. There are a maximum number of available slots of AOS-instances per server for Dynamics AX 4 and at Annata they have reached that maximum. The solution is to add more physical servers and as a result of this the need for physical hard disk space and computing power increases with the number of environments.

The major issue with the infrastructure is the vast number of development environments they need when maintaining their products. On top of the development environments they have release environments, where combinations of development environments are compiled. As an example one can mention their two components called Application Platform (APL) and the Business Platform (BPL). Their major product, IDMS, is built on APL and BPL. They ship all as separate products. In some customer scenarios they bundle IDMS and Connectivity Studio (CNT) from To-Increase. This means one release environment for APL, one for APL and BPL, one for IDMS (with APL and BPL) and yet another one with IDMS with CNT (and APL+BPL). All these different environments require a lot of hardware and a lot of storage.

One suggested solution for Dynamics AX is to use one layer for all solutions, use nightly builds and keep track of the versions produced. Microsoft supplies the full setup of Dynamics AX and limit functionality with license keys. This seems to be logical and sound, and when Annata was asked why they do not use this approach, which would definitely reduce the need for the many environments, the answer was twofold:

1. First of all, there is no way of licensing ISVs solutions like they do at Microsoft. Microsoft does not supply the use of license keys for ISVs. This means that all solutions would be available to everyone, which is not acceptable.
2. Secondly, not all solutions could be treated as the Microsoft certified solutions Annata provide. The rigorous testing, documentation and quality requirements would be too costly to apply to all parts of all solutions.

Annata have more than 50 customers and want to keep them in-house as well, something that is not possible due to these limitations; just a handful of them are kept in-house at the moment.

To facilitate and keep track of builds and versions Annata has developed their own system in Dynamics AX. Identifying and removing elements that have been deleted is one of the shortcomings of the current delivery model that they try to handle with clever tools. With the version control system in Dynamics AX 2009 the means to handle a delivery has been improved but many features are still missing.

3.5.2.1.2 Development

One of the major issues with Dynamics AX is the use of IDs for certain elements, such as tables and table fields. The IDs are used internally to reference these

elements. When changing an ID on a table and synchronizing the application data, the table is dropped from the database and then recreated. If not handled correctly by the developers, the result is potential data loss for customers. Annata aims to use the same IDs in all environments, from development and release environments, and at customer sites.

Internally they use an ID-server, and when they have dependent products those are installed in the lower layers. The development is taking place in the USR-layer. For task management they use a fairly flat system without sub-tasks, etc. and find it easy to work with. The building of the releases and the involved work is time consuming, and they would like to automate the steps to a higher degree.

3.5.2.2 To-Increase

3.5.2.2.1 Infrastructure

To be able to develop, test and deliver each component separately, To-Increase have one or more installations of Dynamics AX per component and version of Dynamics AX. This means a huge amount of installations. Some service packs and versions have compatibility issues and there is also a limit to how many servers you can run at the same time.

Each component should normally have a development, test and release environment. For Dynamics AX 2009 they have a separate server hosting in total 7 environments (4 development, 2 test and 1 release). For the oldest Dynamics AX 3 there exist approx. 50 environments in total, spanning 4 service packs. In AX4, there are in total 15 environments. It is difficult to have data transferred between the environments and the manual labor involved is very time consuming. Each of the servers can be accessed using remote desktop connections.

In many cases, different versions cannot co-exist on the same servers. This implies the need for more hardware. Hardware requirements for running Dynamics AX have increased constantly. Since each component more or less requires at least 3 environments (dev/test/release) it makes it important to know exactly what source code is where and when. This was considered the single biggest issue, since it is hard to maintain all the different environments, checking and cross-checking if elements are identical in two systems is tedious. Creating a good release and bringing everything together takes a lot of time.

3.5.2.2.2 Development

In general To-Increase find themselves in a fairly good situation as they have separate environments for different solutions. For pure development the

environments are easy to work with. The issues arise when the code is to be moved between different environments and errors are to be reproduced. In certain situations fixes may even have to be developed and applied in the test environment, where the issue can be reproduced, before being merged back to the development environment. The product team experiences problems communicating about what is happening in different environments due to the issues with moving both code and data between environments.

Some solutions, like IEM and RCM, are built in the VAR/CUS-layer with BUS-layer containing related products like e.g. the APP. DevCenter is installed in the USR-layer.

Something To-Increase say they lack is the bigger picture, both for elements across different environments and the developer perspective, but also from a product manager perspective. It is hard to see what is going on in different environments and get an overview. In these situations, Excel spreadsheets come into play. The use of spreadsheets undermines the tools within the application that are supposed to be used.

3.5.3 Time consuming activities

3.5.3.1 Annata

Annata identifies a few activities as time consuming. Building new functionality in general is time consuming, with the related activities. When something has been modified proper tests have to be executed. At the moment tests are based on the knowledge of the testers. They want to move towards automated tests, e.g. unit tests.

Keeping documentation up to date in relation to modifications and new features is considered time consuming. The releasing of new versions, compiling everything, doing quality checks and best practice compliance checks, etc. do take a lot of time.

There are also dependencies between different products. A hotfix required for the IDMS solution may very well be related to the application platform or separate stand-alone products maintained in another environments. The hotfix request therefore issues a modification need in another environment. Corrections should be done in the correct environment, progress through test environment and later be released. That release in turn should be implemented in the IDMS development environment, before tests can be executed and a new release can be created.

3.5.3.2 To-Increase

Adding documentation and merging is time consuming. One change implicates updates to training material, documentation and information to the customer.

When issues are reported, a lot of time is spent on double and triple check if the environments are the same. The issues surface in different environments, and the personnel experience different behavior depending on where the issue is investigated. Every time issues like these arise, the different environments have to be compared. Part of the problem is the version management and versions between products. When new tasks or updates have been applied, the version has to be updated as well.

Since there are so many different environments, installing and handling them is taking time. The product development and maintenance of released versions is spanning many major versions of Dynamics AX. The releasing process is also time-consuming. Creating a solid release and putting everything together requires a lot of work.

The lack of time needed to document things properly and the lack of specifications is also a major factor for time being wasted. It may not be every day of the working week, but when it happens it can mean that weeks of work are partially or entirely invalidated. If To-Increase could spend more time designing before they build things, a lot of time could be saved according to the interview subjects.

3.5.4 Improvements

3.5.4.1 Annata

When asked about what they would like to improve a few things come up. One of the things mentioned is the possibility to automate the build and release process. Some kind of tool in the standard system to help the ISVs deliver solutions, release code and deliver it to the customer application. As it is now, it is mostly handled using import/export of xpo-files. Support for semi- or fully automated creation of layers, with compilation etc. would be beneficial.

Annata would also like to go deeper into task management. They are worried about documentation not being done properly or even forgotten. One way they have considered is to start with not only development tasks but also documentation tasks. They do not want to stop development tasks being closed, thus why they need separate documentation tasks. Testing should be part of the development task, a development task should not be closed unless tests are okay. Automated tests, like

e.g. unit tests, and better test scripts are other areas where Annata would like to improve.

As of now, a lot of attention is focused on the IDs in the applications. They do not want to the take IDs into consideration at all, if possible.

3.5.4.2 To-Increase

To-Increase lists a few of the improvements they wish to see. One of the areas is involving information. For example if there is a new release built on a certain product, they would like the stakeholders to be informed, such as the sales people, people responsible for customers, etc. They want all this information to be available in one place, in some kind of centralized environment. They also would like to get e.g. task management into a centralized environment. The interview subject also describes this as a way to give them a forest view of products, environments, tasks, etc. instead of a per-tree-view as they have today.

Creating builds and building versions should have as few steps as possible. The risk of users making mistakes during the process should be minimized by automating the build process.

To-Increase also mentions structured tests, like regression tests, unit tests, etc. as something they would like to do more of.

Another area is the capacity within the company, capacity to do things properly instead of trying to save time by not doing proven best practices (which is not really saving time) according to the interview subject.

Chapter 4
Analysis

In this chapter a discussion and analysis in relation to the results and the Software Configuration Management theories are presented.

4.1 Current problem areas in relation to SCM activities

Neither of the companies have the option to deliver a complete application using a license to limit functionality, like Microsoft do with Dynamics AX, simply due to the fact that such an option is not available to the ISVs; solutions cannot be licensed and controlled properly. This effectively means that all their components and sub-products have to reside in different environments and later be combined/compiled into certain offers or products. This leads to a vast number of environments. These environments are in turn interrelated: both companies express issues with *status accounting*. It is hard for them to know if the latest fixes have been applied to a certain environment. The same version as the main branch of a component may be installed as a sub-component of another product. There may however be fixes applied in the original system, and those have not yet been properly released and therefore the version number has not yet been updated.

The many environments are all under version control when it comes to the elements (source code). However, requirements, tasks, versions, etc. are not. This fits the Developer-Software CM AntiPattern, the Requirements Jeopardy AntiPattern as well as the Decentralized CM AntiPattern. According to Berlack (1992) and Pressman (2005), change control needs to be applied to more than just source code. The current way of working makes it impossible to know if a certain task has been opened, closed, re-opened, etc. or modified in any way. The *identification* of configuration items is lacking as well as the *change control* to these items. Change control, as suggested by Berlack in the change flow diagram in figure 1, chapter 2.4.3.2, is being performed to some extent in both companies. It might not be as strict and structured, but change requests, new features and support issues are all analyzed, reviewed and evaluated before implementation. From what I learned during the interviews, these steps are not that formalized and is sometimes even skipped. The outcome of these change control meetings should be documented and be part of the configuration management.

In relation to the releases, both companies describe a time consuming activity of putting different parts together, ensuring that documentation, training material, included components, etc. are all in order. This work could be interpreted as being part of the configuration audit, even though it is more of a collecting and putting parts together than it is a review activity. However, many of the activities lack support from the system and rely on manual labour. Also, proper and structured testing activities are not clearly visible or traceable with the way the companies work. *Configuration audit* can be improved.

Both companies describe similar structures in the way they work. To communicate with the customers, a separate web application is used. This involves transferring the issues reported in the web application to Development Center, and later updating the same web application again. The many applications each have a separate installation of the tool Development Center for maintaining tasks. This makes each development environment a separate repository, with no possibility of sharing data. This is a major drawback with the current tool and setup since the environments are interdependent. According to Royce (1998), one key practice to improve overall software quality is providing integrated life-cycle environments that support early and continuous configuration control. The tools used are per environment and are therefore not providing the well needed integration.

In summary:

- Missing software configuration management routines, no plan
- Lack of system support for development and project management
- Many products to develop, maintain and support
- One installation per product, version, and service pack resulting in a vast number of environments
- Dependencies between products and/or componentized development

4.2 The SCM plan

I would suggest a proper SCM plan is created with basis from the theories in Chapter 2 Software Configuration Management. The SCM organization has to be described and the level of configuration management for each involved product/component should be stated. This is important since the level differs between products; the certified solutions require more ceremony and better control than the smaller components. The existing support and test organizations should also be described and how they interact with the development. This addresses the SCM Organization Pattern.

A formal description of how the DevCenter tool should be used together with routines for each of the configuration items should be documented. This mitigates the Silver Bullet AntiPattern – a tool alone will not solve all SCM needs. Routines are needed.

4.3 Configuration items

It may be difficult to decide the proper level and definition of configuration item, CI, in the context of Dynamics AX. The definition of CI as described in chapter 2.2 suggests it is a stand-alone, test-alone and use-alone element. In the military standard described, it is any discrete portion of the software, which satisfies an end use function and is designated to configuration management. If configuration management is considered change control in this context, it can be concluded that the elements within Dynamics AX, the building blocks, like classes, tables, forms, etc. should be under change control. This would motivate defining each element as a CI. Uncontrolled change to any of the elements will put the entire system at risk.

As of now, change control is not properly applied to the requirements, specifications, etc. They are also treated a bit different, depending on the source for the needed change.

Both companies register support issues via a web interface. The issues may sometimes get prioritized differently depending on the urgency. As Berlack points out in the SCM activity Identification (chapter 2.4.3.1), capturing specifications and documents related to the software and putting them under configuration management should be done. Not all the support issues will reach development, but it is still important to document each, track any changes to them and possibly be able to trace changes to elements back to the originating issue. Therefore, each support issue should be under configuration management. The handling of change requests and new features is somewhat more formal in both companies from what we learned but needs to be treated as configuration items. In figure 5 below I call them requirements, thus each requirement should be a CI with a classification of its origin.

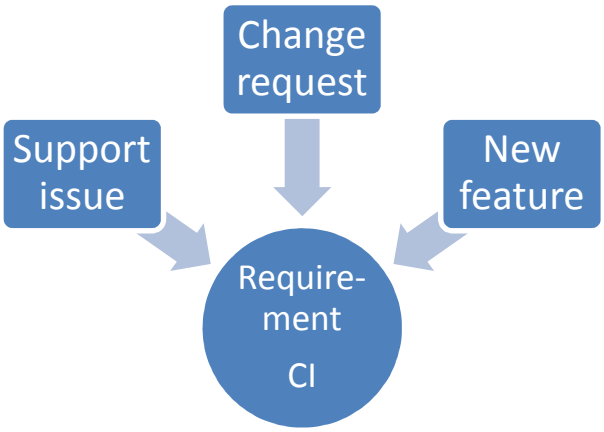


Figure 5 Requirements

The requirement in turn needs to be associated to the correct product. The product has different versions and may contain different features. It may also consist of other sub-products.

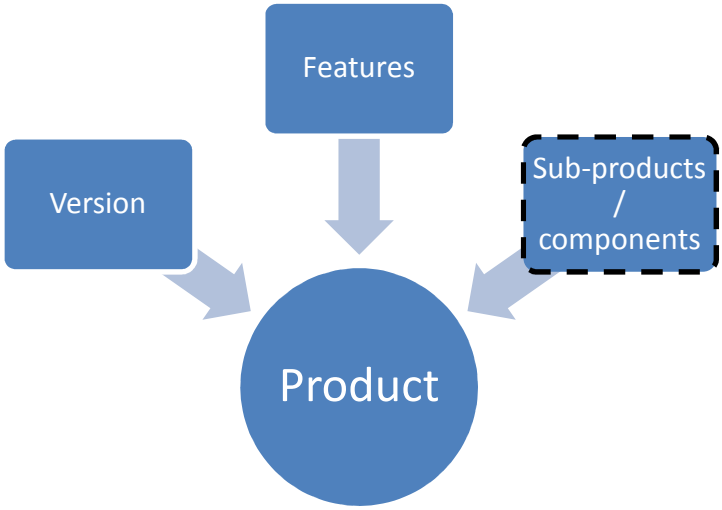


Figure 6 Products

A requirement should then be associated not only to a product, but also to a version and a feature. The product, version and feature should all be under configuration management and considered as informational CIs.

If we map this to the functionality requirements as outlined by Dart, products, versions, features, and requirements will be the components.

The actual elements are as discussed earlier also CIs and they are building blocks of the product. Therefore elements need to be associated to the product. This can be done directly by stating that the product consists of certain elements. Elements are subject to change, which originates from the requirements (support issue, change request or new feature). A requirement may lead to certain activities or tasks. A task can be defined as the work item during implementation of the requirement. If each

element being modified in relation to a task is recorded, the setup will support traceability bottom up and top down as suggested by Berlack. The CIs will have a parent-child relation:

Figure 7 Configuration items and relations

With relations between products, sub-products and the components (version and features), the setup offers a representation of the structure.

Each of these CIs should also be under change control. The amount of routine surrounding each CI must be specified in the SCM plan.

Introducing these CIs will help address the Object-Oriented CM AntiPattern – not only code level but richer hierarchies such as features (components) are required.

4.4 Accounting and auditing

By representing the configuration items as data structures with unique indexes we can construct queries and be able to see data such as:

- All elements related to a product
- All requirements (or tasks) related to single element
- All elements affected by a requirement
- Features in a specific version
- All requirements needed for a specific feature
- Conflicting elements in two products
- Changes to elements between versions

By adding additional data to the configuration items, such as status, the setup will get functionality to support accounting as defined by Dart. It will be possible to extract, review and analyze data such as:

- How many new requirements have been added?
- How many support issues, change requests or new features have been reported?
- What features have issued the most support requests?
- What requirements have been approved but not yet acted upon?
- What requirements are parts of the next (planned) version?

As this information is to be treated as CIs, I would suggest adding a check out/check in functionality to each of them and keeping a history of each modification. This way, all changes will be traceable – from the individual elements, via tasks, requirements, features, and versions up to the product itself. Auditing will be facilitated with this functionality.

4.5 Controlling access

Since the DevCenter is developed in Dynamics AX, it is possible to assign different access rights for users to different entities. This would aid controlling only authorized personnel modify certain categories of CIs, like e.g. and approve/reject. requirements in relation to a feature in the upcoming version.

4.6 Process

Basic functionality to support the management of how the product evolves can be easily implemented. With the CIs as described in chapter 4.3, it is a matter of assigning requirements and tasks to persons responsible, having routines (different strict depending on what is defined in the SCM plan) for reviewing and processing these CIs through different statuses. Adding due dates to requirements and tasks will ease planning and management. Due dates could also be added to e.g. product versions.

4.7 Construction

Releasing a product has been tedious and time consuming for the companies in the study. With the concept of products, features, etc. down to each version of an element, it should be possible to gather all the elements for a certain product. If the assumption is made that only the current active version is to be built and that future work has not yet commenced, then these elements are the ones available in the Dynamics AX application itself. A project in Dynamics AX can be created with these elements and an XPO can be exported. It will also be possible to extract the AOD-files needed. If additional documents and/or descriptions are available for the version, feature, requirement or task, these can also be collected automatically from the system.

Releasing older versions is possible as well, but then a different approach is needed. Two main strategies are possible. Either keeping track of what has been released previously and simply copying those artifacts, or finding what version of each element was associated to the specific product version about to be re-released. Within Dynamics AX there is no such tool available out of the box, but there are external tools to combine xpo-files into one single xpo.

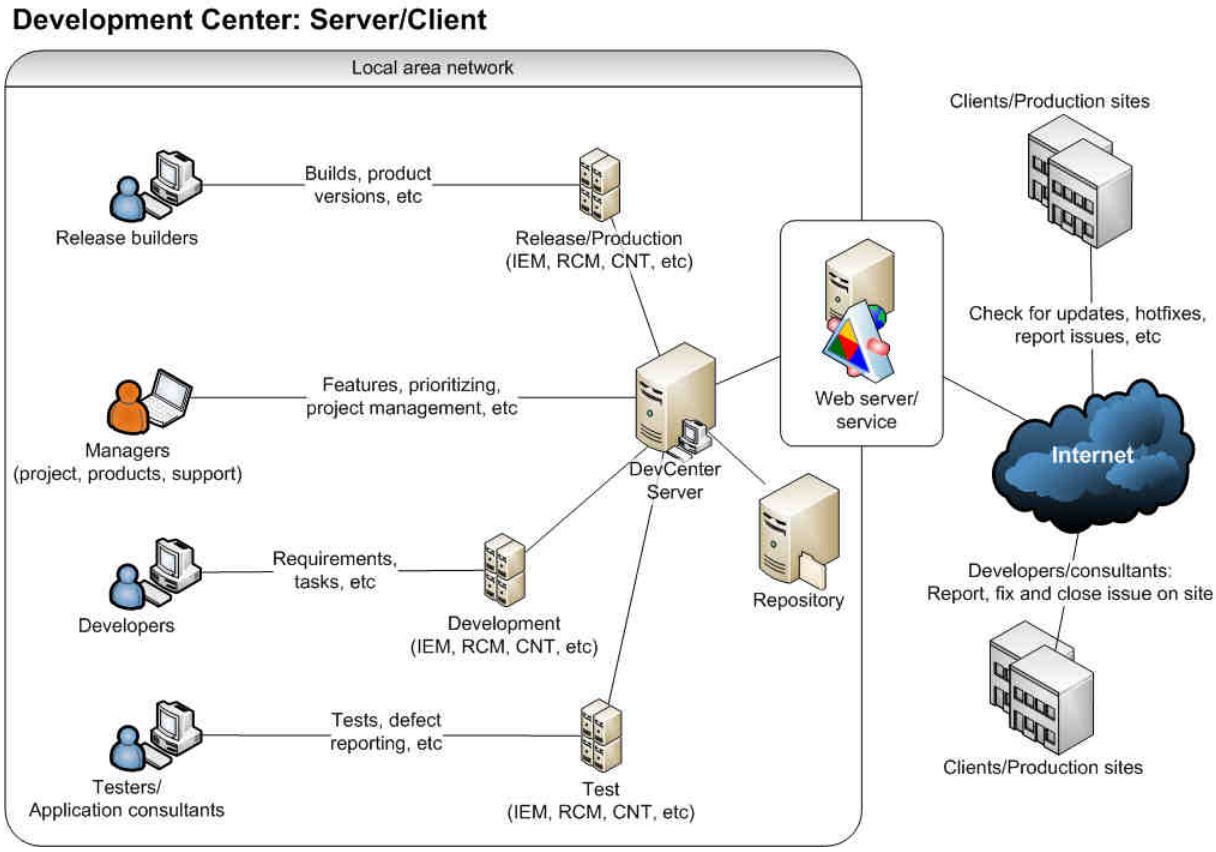
Since AOD-files are often a preferred method for delivering solutions, separate build environments are needed or the AOD-file must be extracted from the current development environment during the release process. If the development environment is used, then re-releasing is not possible and only the latest (current) version can be released.

4.8 How to tackle multiple applications

The multiple applications are something that cannot be avoided easily for the companies in this study. The single most beneficial change to the way they work would be to make a common SCM plan and use a global tool with a central repository. That also addresses Decentralized CM AntiPattern.

I suggest a server/client approach for DevCenter, where each Dynamics AX environment can function as a server or client, depending on setup. Normally only one server should be present. There are tools within Dynamics AX to allow external components to communicate with the application, such as the .NET-connector and support for web services. By using these tools within Dynamics AX, it is possible to allow the application itself to communicate with other Dynamics AX applications.

Since many of the basic functionality requirements for an SCM tool can be made available without too much effort within DevCenter as discussed above, the single most important feature would be to gather all data in one single repository. This would give a well needed complete overview of all environments, the versions installed, the products maintained, requirements, tasks, and elements. It would also allow for communication between clients/production sites and the DevCenter server and central repository.


Development Center: Server/Client
Local area network
Release builders
Builds, product versions, etc
Release/Production (IEM, RCM, CNT, etc)
Clients/Production sites
Check for updates, hotfixes, report issues, etc
Managers (project, products, support)
Features, prioritizing, project management, etc
DevCenter Server
Web server/ service
Internet
Developers
Requirements, tasks, etc
Development (IEM, RCM, CNT, etc)
Repository
Developers/consultants: Report, fix and close issue on site
Testers/ Application consultants
Tests, defect reporting, etc
Test (IEM, RCM, CNT, etc)
Clients/Production sites

Chapter 5
Prototype for DevCenter Server/Client

Making the DevCenter tool a Server/Client application with a central repository could help solve many of the major issues Annata and To-Increase face with their current tools and routines. It is also the most challenging technical part and this chapter describes a prototyped possible solution. The prototype was implemented and offered a fully functioning DevCenter Sever.

Support for software configuration management and increased control over the processes involved when executing projects and/or product development is a goal. A central repository is needed where everything can be gathered and made available to others. Support for a process spanning multiple applications is required.

The preconditions are to avoid using sandboxes and having to rely on external server technologies: also, the overhead must be kept to a minimum.

The prototype is a proof of concept. It is using Dynamics AX's built in functionality to accomplish a centralized environment across multiple applications.

5.1 Background

DevCenter has been extended and now includes:

- Products
- Versions of products
- Product features per product version
- Requirements for the product version and feature
- Tasks for requirements

Elements (tables, classes, etc) that developers check out/in for editing are stored and can be associated to tasks. It will thereby also be associated to product versions via the requirements and it is possible to track what element is in what product version. Versions of elements are stored when checking out/in.

5.2 Software configuration management

Software configuration management (SCM) is an umbrella activity and something that should be part of the entire lifecycle.

SCM activities should be applied to elements, tasks, requirements, features, products, and releases by default. These areas and their associated artefacts should be possible

to identify, control change to, perform status accounting on, and also to perform configuration audits on. The level of control should however be optional.

5.2.1 Tracking needs

Below is a list of what the system needs to track and control changes to, and why:

- Releases – product versions grouped together make up a deliverable. What releases exist and what do they consist of?
- Product versions – what features can be found in what product version?
- Features – have the features for a product changed?
- Requirements – changed requirements have a huge impact on the product.
- Tasks – changed tasks have a huge impact on the product.
- Elements – these are the smallest building blocks (classes, tables, forms, etc.) and changes to elements change the behavior of the product.

Knowing that a release consists of a product that has a feature with certain requirements, and that the requirements have tasks with certain versions of different elements, it is possible to track all behaviors both top-down and bottom-up. When changes to this chain are controlled, considered and approved before being implemented, the overall product quality can improve and issues can be resolved more efficiently. By marking any of the items in the chain as "touched" or "modified" when changes are introduced, areas that need documentation, tests, or any other review as part of the audit can be easily identified.

5.3 Assumptions

Some assumptions have been made to realize the first prototype of the DevCenter Server:

- It should use a single repository.
- All is taking place within a local area network, for now at least (see next point).
- File shares should reside in the repository as well, not only the "data"; all folders and files must be available in the entire setup.
- A single development environment on a server level per product (i.e. no sandboxes per client for individual developers).
- When a product version is released, it is stopped from further development and is treated as a baseline. The only allowed changes to a released version are hot-fixes.
- Before each new build, previous hot-fixes must be taken into account and merged to the "mainline", i.e. to the new build.

- Hot-fixes that have not been merged must be highlighted before a new version is built so that a decision can be made whether or not to include them.
- Each build must be uniquely identifiable and the ID of the build must be in the AOT.
- Each hot-fix applied to a system must be uniquely identifiable and the ids of the fixes must be in the AOT.
- If a product consists of several components, to which modifications are allowed for integration purposes, modification of the components should be treated as part of the major product.
- If a product consists of several components, to which improvements are made, they should be treated as hot-fixes and thus be added to the original component's hot-fix list.
- Components and products are grouped together via releases.
- A global user ID must be available for tracking actions per user.
- To allow working on a task from several different environments, an exclusive lock is required or merging capabilities. For now, exclusive locks with check in/check out are suggested.

5.4 Technical solution

This section describes the design and ideas behind the development.

5.4.1 General idea

KISS – Keep It Simple Stupid – is a good strategy that we have tried to enforce.

5.4.1.1 Server/client elements

To make implementation easy, the server and client contain the exact same tables and forms. Some functionality will however not be available on the client, and additional functionality may be needed on the server for the future.

For now, a parameter specifying if it is a server or client decides the behavior.

The decision to have the same tables and forms is not a mandatory requirement and can be changed later.

5.4.1.2 Data

All data should be on the server. Clients should be able to connect to the server and modify the data. Data is available on the server and you are allowed to change it, but all data in your client session is volatile unless you save it to the server.

5.4.1.2.1 Client data in temporary tables

Aligned with the web server/client metaphor, the client only operates on data stored in tables made temporary in Dynamics AX.

5.4.1.3 Basic interface

The interfaces are based on some ideas taken from a repository pattern.

5.4.1.4 Communication

The communication between server and client has been implemented to use two different technologies, but are basically the same.

5.4.1.4.1 Web service

A web service has been created. It has tree operations that use the .NET Business Connector to logon to AX and retrieve the data asked for.

5.4.1.4.2 .NET Business Connector

To speed up communication in a local area network the .NET Business Connector can be used directly. It uses the same operations as the web service.

5.4.1.5 Session

On the client, a session is initialized when a request for certain tables is done. The session is implemented as a singleton and downloads all involved tables from the server, keeps a reference to these tables made temporary by code and offers some basic functionality for communication with the server.

5.4.2 Web service

The web service was created using C#.NET and it offers three operations:

1. CallStaticClassMethod:
    a. Input:
        i. ClassName as string
        ii. methodName as string
    b. Output:
        i. A string with xml. *Comment*: this is a bad design, it should just return the xml, but it was a corner cut with data types in AX, .NET and the .NET Business Connector for AX.
2. CallStaticClassMethodParm1
    a. Input:
        i. ClassName as string
        ii. methodName as string

iii. parm as string

b. Output:

i. A string with xml (see earlier comment)

3. CallStaticClassMethodParm2

a. Input:

i. ClassName as string

ii. methodName as string

iii. parm1 as string

iv. parm2 as string

b. Output:

i. A string with xml (see earlier comment)

All these operations call the static method in the specified class with the parameters specified. It uses the .NET Business Connector to do this.

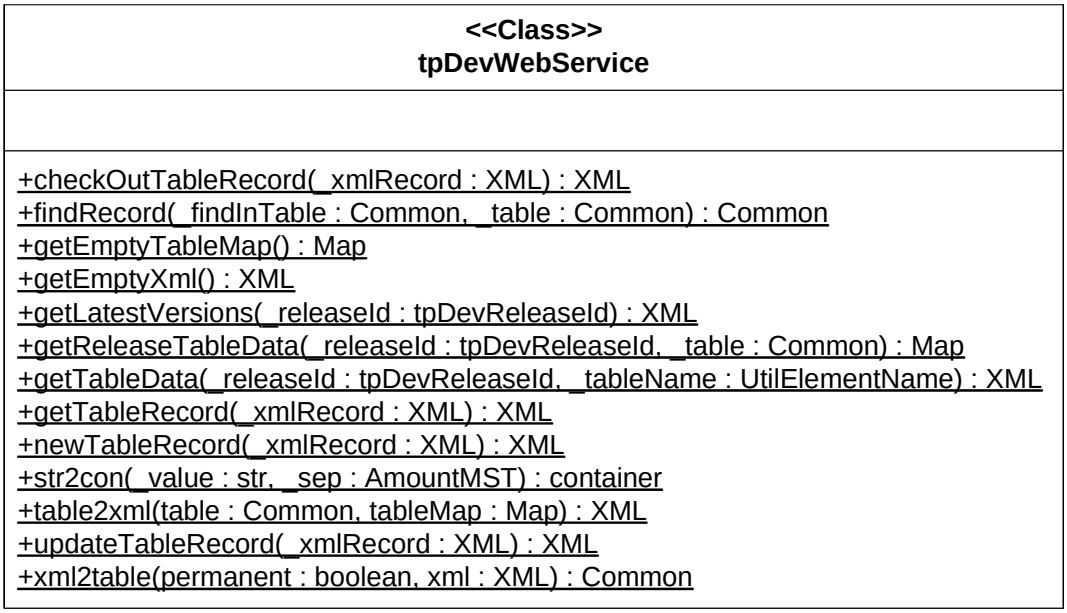
There are security issues with this opening and a proper interface should be implemented instead of exposing all classes and static methods.

5.4.3 The web service class in AX – tpDevWebService

This class offers all of the operations that are used via the web service and, if setup to do so, the .NET Business Connector. Only the methods that return XML are used in the communication.

5.4.3.1 UML and method descriptions

Below is a UML class describing the tpDevWebService class.


<<Class>>
tpDevWebService
+checkOutTableRecord(_xmlRecord : XML) : XML
+findRecord(_findInTable : Common, _table : Common) : Common
+getEmptyTableMap() : Map
+getEmptyXml() : XML
+getLatestVersions(_releaseId : tpDevReleaseId) : XML
+getReleaseTableData(_releaseId : tpDevReleaseId, _table : Common) : Map
+getTableData(_releaseId : tpDevReleaseId, _tableName : UtilElementName) : XML
+getTableRecord(_xmlRecord : XML) : XML
+newTableRecord(_xmlRecord : XML) : XML
+str2con(_value : str, _sep : AmountMST) : container
+table2xml(table : Common, tableMap : Map) : XML
+updateTableRecord(_xmlRecord : XML) : XML
+xml2table(permanent : boolean, xml : XML) : Common

Comments:

- Method: *checkOutTableRecord* - Attempts to check out a record sent in as XML. Transforms the xml to record with `xml2table` method. Calls `findRecord` with the transformed record. Checks the found record, if any, and assumes the record has a field called "CheckedOutTo" and whether that field is empty or not. If it is empty, the found record is updated to match the input record. If a record was found, that record is returned. Otherwise and empty xml is returned by calling `getEmptyXml`.
- Method: *findRecord* - Tries to find the record in common _table in the data. It's possible to specify a `_findInTable` that will allow searches in a reference to a temporary table. The method builds up a query based on the `firstUniqueIndex` of the table sent in. For a `_tableRecord` to be found, it must have a unique index setup as well as a `recid`. The record returned is selected for update.
- Method: *getEmptyTableMap* - Returns a map where `RecId` and the `Record` can be stored. Used only to ease up the creation of the map needed from several places.
- Method: *getEmptyXml* – Returns an empty xml that still validates. Has a dummy element:

```
<?xml version="1.0" encoding="utf-8"?>
<dummy />
```

- Method: *getLatestVersions* – Returns an xml with all the latest released versions of the products that are associated to the `_releaseId` input:

```
<?xml version="1.0" encoding="utf-8"?>
<text>
    <version product="Cnt">Ver 2</version>
</text>
```

- Method: *getReleaseTableData* - Retrieves a set or subset of the data in the table specified by the common `_table` given the `_releaseId` as the common denominator. Some tables are returned in total, others that are related to a product version get just the data associated to the specific product version. If a product version for the `_releaseId` is set as active however, it is assumed it is the development environment and returns all versions for that product. Returns a map created by `getEmptyTableMap` containing the recids and records found.
- Method: *getTableData* - Returns an xml with data in the table specified by the `_tableName`. The `_releaseId` is optional, and if provided this method calls `getReleaseTableData` to get a subset of the data related to `_releaseId`. This method calls `getEmptyTableMap` and inserts the found records in the table map. The `table2xml` is then called with the table map.

- Method: *getTableRecord* - Transforms the input `_xmlRecord` to a record with `xml2table`. It then tries to find the specified record using the method `findRecord`. If a table record is found, it is converted to xml with `table2xml`. If it is not found, an empty xml is returned with `getEmptyXml`.
- Method: *newTableRecord* - Converts the input `_xmlRecord` to a common with `xml2table` and tries to find that record using `findRecord`. If a recid is found for that common, i.e. an existing record was found, and a new record must be created. Depending on the table, certain initializations are done. If a recid was not found by the `findRecord`, we assume it is correct from the `_xmlRecord` and just insert it. The resulting record is transformed to xml again by `table2xml`. That xml is returned. If no table record was found nor created, an empty xml from `getEmptyXml` is returned.
- Method: *str2con* - Modified from the `global` class. Original does not take into account the length of the separator, this method does however.
- Method: *table2xml* –Takes a record in as common table. The `tableMap` is optional and if it is the default value, the record currently in table is inserted. Thereafter the map enumerator is used to loop over all the records, creating an xml out of it. The resulting xml is returned. Can also handle arrays like dimensions.

```
<?xml version="1.0" encoding="utf-8"?>
<Records>
     <DevProduct RecId="5637145359">
                <ProductId>IEM</ProductId>
                <Name>Product Name 2</Name>
                <Prefix></Prefix>
                <LabelFileId></LabelFileId>
                <ConfigurationKey></ConfigurationKey>
                <Path></Path>
                <Description></Description>
                <Layer>usr</Layer>
                <SecurityKey></SecurityKey>
                <FunctionalOwner></FunctionalOwner>
                <TechnicalOwner></TechnicalOwner>
                </DevProduct>
     <DevProduct RecId="5637145360">
                <ProductId>RCM</ProductId>
                <Name>Product Name 3</Name>
                <Prefix></Prefix>
                <LabelFileId></LabelFileId>
                <ConfigurationKey></ConfigurationKey>
                <Path></Path>
                <Description></Description>
                <Layer>usr</Layer>
                <SecurityKey></SecurityKey>
                <FunctionalOwner></FunctionalOwner>
                <TechnicalOwner></TechnicalOwner>
     </DevProduct>
</Records>
```

- Method: *updateTableRecord* - Takes a `_xmlRecord` as input and transforms it to a common with `xml2table`. It then tries to find the record using `findRecord` and the result is called `forUpdateRecord`. If the result has a recid, the `_xmlRecord` data is copied to the `forUpdateRecord` which then is updated. If the `forUpdateRecord` has a recid, it is transformed to xml by `table2xml` and the result returned. Otherwise an empty xml is returned by `getEmptyXml`.
- Method: *xml2table* - Takes an xml table as input and loops over the records node of the xml. It builds up the record from the xml and then inserts it into a temporary table. That temporary table is then returned. The parameter boolean `permanent` which defaults to `false` it no longer in use, but could be used to skip the temporary table and have the records from the xml inserted permanently.

5.4.4 The Session – tpDevSession

This section describes the session class, implemented as a singleton using the global cache. Since all references go via the public static instance method, it will be created when first called upon. When that happens, it initializes the Dev* tables and keep a reference to them during its lifetime.

The CLRObject dynAx is only used if it is set up to use the .NET Business Connector directly; otherwise the communication goes via web services.

5.4.4.1 UML and method descriptions

Below is a UML class describing the tpDevSession class.

<<Class>> **tpDevSession**
#devVersionMilestone : DevVersionMilestone #devVersionHotfix : DevVersionHotfix #devVersionFile : DevVersionFile #devVersionFeatureTest : DevVersionFeatureTest #devVersionFeature : DevVersionFeature #devVersionElement : DevVersionElement #devVersion : DevVersion #devUserTask : DevUserTask #devUser : DevUser #devTaskElement : DevTaskElement #devTask : DevTask #devRequirement : DevRequirement #devProduct : DevProduct #devParameters : DevParameters #devHistory : DevHistory #devElementVersion : DevElementVersion #devElement : DevElement #devAttachment : DevAttachment #dynAx : CLRObject
+disposeInstance() +finalize() +getDynAx() : CLRObject +getTable(_table : Common) : Common +init() +insertTableRecord(_tableRecord : Common) : Common +instance() : tpDevSession -new() +setTable(_newTableData : Common) : Common +updateData(_table : Common) +updateTableRecord(_tableRecord : Common) : Common

Comments:

- Method: *disposeInstance* - Disposes of the instance by finalizing it and removing the reference in the global cache.
- Method: *finalize* - Removes the references to the temporary tables and logs off if the business connector has been used.
- Method: *getDynAx* - Returns the .NET Business Connector member variable. Is `null` if it is not setup to use the connector.
- Method: *getTable* - Returns the reference to the member table specified by the type of the common use .
- Method: *init* - Logs on to AX if it is setup to use the .NET Business connector. Calls to get data from the server for each member table.
- Method: *insertTableRecord* - Sends a request to the server to insert a record. Returns the record that the server sends in the reply.

- Method: *instance* - Creates an instance of the class and ensures that only one instance can exist.
- Method: *new* – Is private, called only from the instance method.
- Method: *setTable* - Sets a member table to temporary and the temporary data to what is sent in via common `_newTableData`. Returns the temporary table.
- Method: *updateData* - Retrieves a whole table from the server, and calls `setTable` with the result as input.
- Method: *updateTableRecord* - Updates the input common `_tableRecord` to the server. Returns the updated table record retrieved from the server.

5.4.5 Allow a form to use temporary tables

This section describes how the forms and tables were transformed to allow operations on temporary tables.

Assume that the server version and client version share the same tables and forms. Working with the server version should be exactly as working with a normal form, directly towards the data in that environment. Thit behavior is expected on the client side as well, but towards data held in the session class (`tpDevSession`).

The implementation of this functionality is not ideal. The code should be rewritten and moved from the forms and put into classes. Some of the macros are used just to allow for less duplication of code with the choices made for the prototype.

5.4.5.1 The table

This section describes changes made to tables in order to implement the temporary table synchronization. In general write methods are modified to make sure the modifications are updated on the server. All code that retrieves a record in any way must be modified. (This applies to all elements in the AOT, not only tables.)

5.4.5.1.1 The table – `update` and `insert`

There are a few things needed on the table level to make sure everything is consistent. The insert and update methods should be modified as follows. The `insert` method:

```
public void insert()
{
    ;
    if(!tpAXNetworkingParameters::find().IsServer)
    {
        this.data(tpDevSession::instance().insertTableRecord(this));
        this.doInsert();
    }
    else
    {
        super();
    }
}
```

The `update` method:

```
public void update()
{
    ;
    if(!tpAXNetworkingParameters::find().IsServer)
    {
        this.data(tpDevSession::instance().updateTableRecord(this));
        this.doUpdate();
    }
    else
    {
        super();
    }
}
```

Doing this allows changes to the tables to be temporary and mirrored to the server regardless of where they occur; in code or in forms.

5.4.5.1.2 The table – `find` and `exists`

The `find` and `exists` methods must also be modified. The reason is that the search must be within the temporary data, otherwise an empty record will be retrieved. The `find` method:

```
static TableName find(  TableNameRecordId _id,
                        boolean selectForUpdate = false)
{
    TableName theRecord;
    ;
    if(!tpAXNetworkingParameters::find().IsServer)
    {
        theRecord = tpDevSession::instance().getTable(theRecord);
    }

    if (_id)
    {
        theRecord.selectForUpdate(selectForUpdate);
        select theRecord
        where theRecord.TableNameRecordId == _id;
    }
    return theRecord;
}
```

The `exists` method:

```
static boolean exists(  TableNameRecordId _id,
                        SysElementName  name)
{
    return _id && TableName::find(_id).recId != 0;
}
```

This will allow for code in different areas to function reasonably well. There might very well be glitches that need patching once the prototype is transforming into something with higher fidelity.

5.4.5.1.3 The table – `validateField`

For some tables it is necessary to overwrite the `validateField` method. The reason for this is that the kernel goes directly to the database to try to validate the field. This forces implementation of validate functionality within the temporary data. Below is an example of how that can be achieved. In this specific example the validation is implemented for the `DevRequirement` table where the `ProductId` is located. The `ProductId` needs to be validated against the `DevProduct` table:

```
public boolean validateField(fieldId _fieldIdToCheck)
{
    boolean ret;
    DevProduct          devProduct;
    ;
    if(!tpAXNetworkingParameters::find().IsServer)
    {
        switch(_fieldIdToCheck)
        {
            case fieldNum(DevRequirement, ProductId) :
            {
                devProduct = tpDevSession::instance().getTable(
                                devProduct);
                select devProduct
                    where devProduct.ProductId == this.ProductId;
                ret = devProduct.RecId != 0;
                break;
            }
        }
    }
    else
    {
        ret = super(_fieldIdToCheck);
    }

    return ret;
}
```

5.4.5.1.4 The table – `lookup`

To enable lookups it is mandatory to write specific lookups, since the kernel goes to the database table to find records for the lookup. Below is an example, once again on the `DevRequirement` table and for the `ProductId`:

```
public void lookup(FormControl _formControl, str _filterStr)
{
    SysTableLookup          sysTableLookup;
    Query                   query;
    QueryBuildDataSource    datasource;

    DevProduct              devProduct;
    ;
    if(!tpAXNetworkingParameters::find().IsServer)
    {
        sysTableLookup = SysTableLookup::newParameters(
                        tableNum(DevProduct),
                        _formControl);

        datasource = query.addDataSource(tableNum(DevProduct));
        query = new Query();

        sysTableLookup.addLookupfield(fieldNum( DevProduct,
                                                ProductId));
        sysTableLookup.addLookupfield(fieldNum( DevProduct,
                                                Name));

        devProduct = tpDevSession::instance().getTable(devProduct);
        sysTableLookup.parmTmpBuffer(devProduct);
        sysTableLookup.parmQuery(query);
        sysTableLookup.performFormLookup();
    }
    else
    {
        super(_formControl, _filterStr);
    }
}
```

5.4.5.2 The form

Forms that are to display temporary data need some modification in order to work properly. In the following sections the modifications and schemes are described.

5.4.5.2.1 The form – display data

On the form, the datasources `init` method should be overwritten:

```
public void init()
{
    if(!tpAXNetworkingParameters::find().IsServer)
    {
        TableName_ds.allowEdit(false);
        TableName.setTmp();
        TableName.setTmpData(
            tpDevSession::instance().getTable(TableName)
            );
    }

    super();
…
```

This says that if it is a client (i.e. not a server) make sure the users are not allowed to do any changes. To do changes, they need to acquire an exclusive lock. The data source is set as temporary and the temporary data is retrieved from the `tpDevSession`.

5.4.5.2.2 The form – allowing editing of data

To allow editing, an exclusive lock of the record must be acquired. This is true regardless of if the table is temporary or not, it is the scheme chosen for the first prototype to allow remote updating of shared records.

For that purpose a `CheckedOutTo` field is created in the physical table. Editing is allowed if that field is set to the current user. Since no notion of a global user exists yet, it was decided to use current user name and the name of the database: `strFmt('%1:%2', curUserId(), systemInfo.getloginDatabase());`

The `active` method of the datasource triggers a method that will check if `CheckedOutTo` is set to the current user and allow editing accordingly. A check out button on the form calls the `checkOutTask` metod:

```
void checkOutTask()
{
    AXWebServiceDevConsumer.Service1    axWebService;
    XML                                 xml;
    ClrObject                           clrException;
    Microsoft.Dynamics.BusinessConnectorNet.Axapta dynAx;
    #macrolib.tpDev
    ;
    try
    {
        #tpCheckOut(devRequirement)

    }
    catch( Exception::CLRError )
    {
        //BP Deviation Documented
        clrException    =   CLRInterop::getLastException();
        if( clrException )
        {
            error(clrException.get_Message());

            clrException = clrException.get_InnerException();
            if (CLRInterop::isNull(clrException) == false)
            {
                error(clrException.get_Message());
            }
        }
    }
    this.doResearch();
}
```

The macro `tpCheckOut` looks like this:

```
#localmacro.tpCheckOut
    startLengthyOperation();
    if(%1.isTmp())
    {
        %1.CheckedOutTo = tpDevTmpTables::checkOutUserName();
        xml = tpDevWebService::table2xml(%1);
        %1.CheckedOutTo = '';

        if(#UseDotNetConnector)
        {
            dynAx = tpDevSession::instance().getDynAx();
            xml = dynAx.CallStaticClassMethod(
                    #tpDevWebService,
                    'checkOutTableRecord',
                    xml);
        }
        else
        {
            axWebService = new AXWebServiceDevConsumer.Service1();
            xml = axWebService.CallStaticClassMethodParm1(
                    #tpDevWebService,
                    'checkOutTableRecord',
                    xml);
        }
        tpDevSession::instance().updateTableRecord(
                    tpDevWebService::xml2table(xml));

        %1.data(tpDevWebService::findRecord(
                    %1,
                    tpDevSession::instance().getTable(%1)));
    }
    else
    {
        %1.CheckedOutTo = tpDevTmpTables::checkOutUserName();
    }
#endmacro
```

The idea is to first set the `CheckedOutTo` and send the record to the server. The server method will make sure it is not checked out on the server side, update the record and then return it. The session data is updated with the record retrieved from the server. That record is then set as the data of the current record on the data source.

5.4.5.2.3 The form – updating data

The `write` method on the data sources must be overridden to allow modification of data to be treated properly. Since the table is already modified to handle the `insert` and `update`, all we need to do is make sure they are called:

```
public void write()
{
    if(TableName.isTmp())
    {
        if(TableName.RecId)
        {
            TableName.update();
        }
        else
        {
            TableName.insert();
        }
    }
    super();
}
```

5.4.5.2.4 The form – finished editing

When the editing of a record is finished, the exclusive lock is released to allow others to access it. This is achieved by a check in button on the form, calling code like this:

```
void checkInTask()
{
    AXWebServiceDevConsumer.Service1    axWebService;

    XML                                 xml;
    ClrObject                           clrException;
    str                                 tmpStr;
    #macrolib.tpDev
    Microsoft.Dynamics.BusinessConnectorNet.Axapta dynAx;
    ;
    try
    {
        #tpCheckIn(devRequirement)
    }
    catch( Exception::CLRError )
    {
        //BP Deviation Documented
        clrException    =   CLRInterop::getLastException();
        if( clrException )
        {
            error(clrException.get_Message());

            clrException = clrException.get_InnerException();
            if (CLRInterop::isNull(clrException) == false)
            {
                error(clrException.get_Message());
            }
        }
    }
    this.doResearch();
}
```

The `tpCheckIn` macro looks like this:

```
#localmacro.tpCheckIn
    startLengthyOperation();
    if(%1.isTmp())
    {
        tmpStr = %1.CheckedOutTo;
        %1.CheckedOutTo = '';
        xml = tpDevWebService::table2xml(%1);
        %1.CheckedOutTo = tmpStr;

        if(#UseDotNetConnector)
        {
            dynAx = tpDevSession::instance().getDynAx();
            xml = dynAx.CallStaticClassMethod(
                                    #tpDevWebService,
                                    'updateTableRecord',
                                    xml);
        }
        else
        {
            axWebService = new AXWebServiceDevConsumer.Service1();
            xml = axWebService.CallStaticClassMethodParm1(
                                    #tpDevWebService,
                                    'updateTableRecord',
                                    xml);
        }

       tpDevSession::instance().updateTableRecord(
                    tpDevWebService::xml2table(xml));

        %1.data(tpDevWebService::findRecord(
                    %1,
                    tpDevSession::instance().getTable(%1)));
    }
    else
    {
        %1.CheckedOutTo = '';
    }

#endmacro
```

This macro stores to whom the record is checked out, removes that text and then creates an xml without the checkout-field set. It then sets the current record as checked out again. The xml is sent to server with a request to update it. The session data is updated with the record retrieved from the server. That record is then set as the data of the current record on the datasource.

5.4.5.2.5 The form – refresh data

On the forms we have added a button to retrieve new data from the server. It sets the table as temporary, updates the data and retrieves data from the server:

```
void clicked()
{
    super();
    TableName.setTmp();
    tpDevSession::instance().updateData(TableName);
    TableName.setTmpData(
        tpDevSession::instance().getTable(TableName));
    element.doResearch();
}
```

The `doResearch` method is simply doing a refresh, but allows for a find in the temporary data:

```
void doResearch()
{
    TableName tableNameTmp;
    ;
    if(DevRequirement.isTmp())
    {
        tableNameTmp = tpDevSession::instance().getTable(
                            tableNameTmp);
    }

    if(tableNameTmp.RecId != tableName.RecId)
    {
        select tableNameTmp
            where tableNameTmp.RecId == tableName.RecId;
    }
    tableName_ds.executeQuery();

    if (tableNameTmp)
        tableName_ds.findRecord(tableNameTmp);
}
```

For some reason this method was sometimes causing troubles. Most likely because the cursor pointed to a certain record and when the select statements were executed the cursor was changed. Therefore a check is needed to see if the recids are different before doing any search. Normally the cursor is at the right position and the result from the `getTable` method is what we want to find.

5.5 Further development

During the development of the prototype some ideas for further development were sprung as well.

5.5.1 Clean up

The code must be cleaned up and streamlined. Gather as much as possible in classes and move code away from forms.

One suggestion could be to use the table and form behavior from the APP classes to make it easy to include the new tables as temporary. This could be implemented separately to make sure it does not interfere with the App product. I would also make sure that the settings and data needed for that was stored in the AOT somehow, to avoid having to rely on data in the client database.

5.5.2 Security

There are major security issues with the current prototype. For example the web service is currently allowing all class static methods to be called. That could be improved by omitting the class name in the calls to make sure only methods of that specific class can be called.

The webservice is run by a user with administrator rights in AX. Multiple options are available to move away from this setup. I performed some tests where the user needs to logon to the website, where identification can be done against the Active Directory, and those credentials used for making the connection via the .NET Business Connector. Unfortunately I didn't find out exactly how the sessions and users are treated in the connector and didn't prioritize to find out. Some findings suggested that the first user to logon is the credentials used until the connection times out. Other users will also logon as that specific user then and code will be executed with a possibly incorrect policy.

Another issue is the data sent back and forth. It is in plain text and can easily be read by eavesdroppers.

There is no input validation and the methods exposed by `tpDevWebService` allow a user to extract information as they like. This is a major security risk.

5.5.3 Sessions

Retrieving data from the server is expensive when measured in time. One of the more time consuming activities is logging on to AX via the .NET Business Connector. Maintaining a session on the website that is logged on to AX improves performance considerably. This could be implemented without too much effort. From a client in a local network setup, the `tpDevSession` class holds an active session to the server via the .NET Business Connector, which is why that setup is much faster at the moment.

5.5.4 Schema for update and refresh of data

All Dev* tables are currently transferred to the client at the first initialization of the session class `tpDevSession`. When a record is checked out, each write is sending the entire record back and forth between the server and the client. The good thing is that data is stored on the server as soon as something is changed, the bad thing is that

data is stored on the server as soon as something is changed; there are both pros and cons for this.

Several different improvements for this schema are possible:

- Don't initialize everything by default; only retrieve data when it is requested. This will result in shorter startup time, but some delays when something new is requested.
- Create separate forms and tables for the client; mimic web pages where the user first gets a list and then clicks a link to go to the details. By doing that it is possible to limit the fields needed for most of the tables, and only retrieve details for individual records when needed.
- Define what fields are really needed on a client level; skip all other fields in the communication.
- Remove updates for each write when modifying data. Only perform an update to the server when the record is checked in.

5.5.5 Data model

At the moment, the data model must be kept in mind when the data is transferred to the client. There is nothing ensuring data completeness so to say. With the ideas from the entity framework in the APP product it is possible to make sure associated records are always grouped together when the client is initialized, keeping the data model intact without missing records.

5.5.6 Exclusive locks

The prototype uses exclusive locks for allowing updates to the shared records. Another alternative could be to implement merging capabilities for individual fields in the table. By comparing the old data before doing a write, one could decide whether anyone else has modified the data. If a field has changed, the user then has to decide what version to use, or how the data should be merged.

5.5.7 Robustness

The communication is not robust at the moment. Error messages must be improved to provide proper feedback and exception handling must be improved. The chain of updates that are triggered automatically must be controlled. For example, when a task is completed changes to the requirement is triggered. The task might be checked out, the requirement might not. This will result in unauthorized updates to the requirement. There are several approaches to this problem. One approach is to always check out records on the top level for certain operations. This will however not make collaboration on the same set of records as flexible as with other approaches.

5.5.8 Elements

Elements are tables, classes, reports, etc. and can be found in the AOT. Elements that are checked out and stored in Dev* tables are at the moment handled as temporary tables. I believe it is a good strategy to try to store this information on the server side as well. The reason for this statement is the increased flexibility with no data dependencies to different clients. Patterns also suggest that de-centralization should be avoided. However, the data model must be modified if that strategy is to be supported.

5.5.9 Configuration management

At the moment, the system does not support some of the desired configuration management routines. Below is a list with suggestions for improvement in this area:

- Create a parameter setup that allows certain user/s to act as a configuration control board.
- Ensure that certain changes are only allowed by members of the configuration control board:
    - Approving requirements.
    - Modifying the content of requirements that are approved.
    - Changing features.
    - Changing products and versions.
- Make sure there is traceability between builds and product releases; a new build is a new product release version.
- Make sure hotfixes are belonging to the version where they originate from and apply to.

5.5.10 Persistent or non-persistent data

At the moment data is non-persistent, it is kept in temporary tables. With the current approach each write to a record issues an update to the server. Therefore the only data a user may lose is the same as when entering information in a web form, the current data not yet saved to the server.

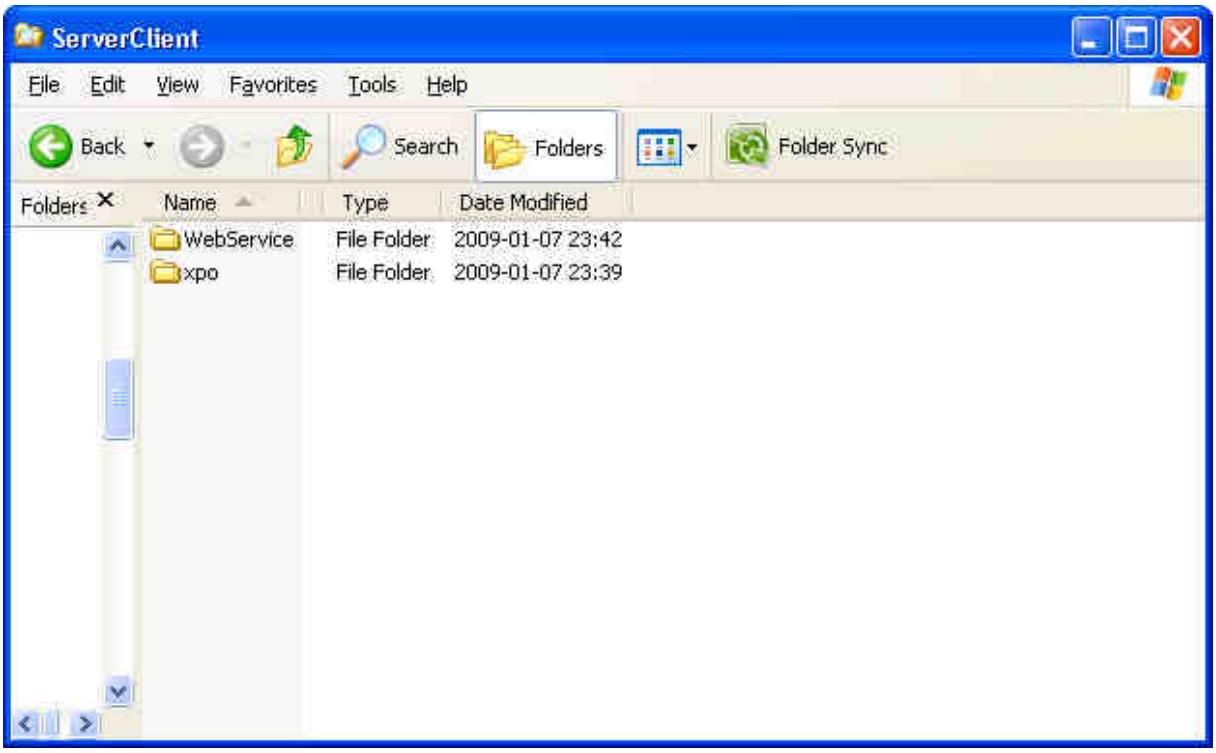
Another possible approach is to keep records stored locally but use the same mechanisms as described above. Keeping data locally was tested and abandoned early in the process, but with another approach. After discussions I implemented a quick demo where data is stored locally as well.

There are issues that arise by moving to a persistent data setup, but a good synchronization pattern can solve most problems. There are of course pros and cons with a temporary table setup as well.

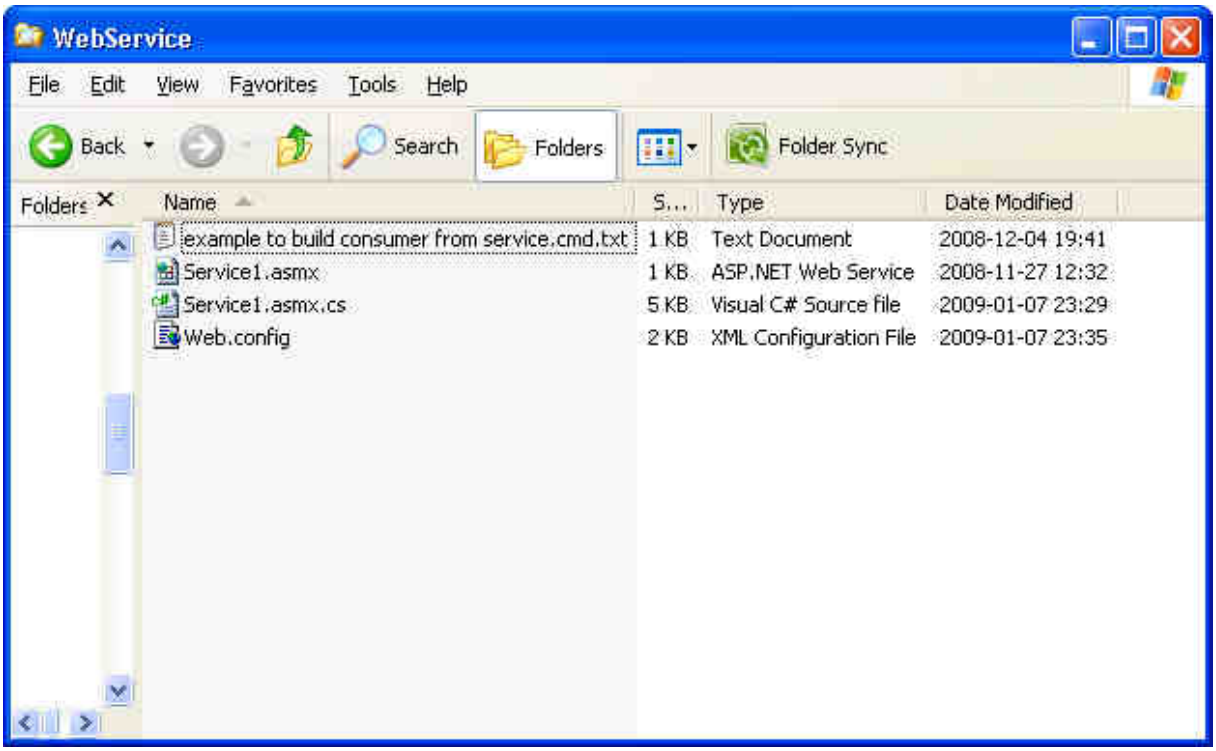
The idea I had for the demo I created was to have a service class running as a batch, updating the tables needed. Checked out records for the specific environment shouldn't be updated since they are currently edited in the local environment. Apart from that, all data could be updated. Although not implemented in the demo, the idea was to keep track of the latest synchronization time in the local environment, it is easy to get only records with a modified time later than last synchronization time. With no synchronization time, everything is synchronized. Deletes from the server must be handled somehow, as well as possible off-line editing from the client. The deletes must be marked somehow, perhaps in a separate table that will be used to synchronize the deleted records to the client.

5.6 Setting the prototype up

In the zip file with the delivery, these folders can be found:

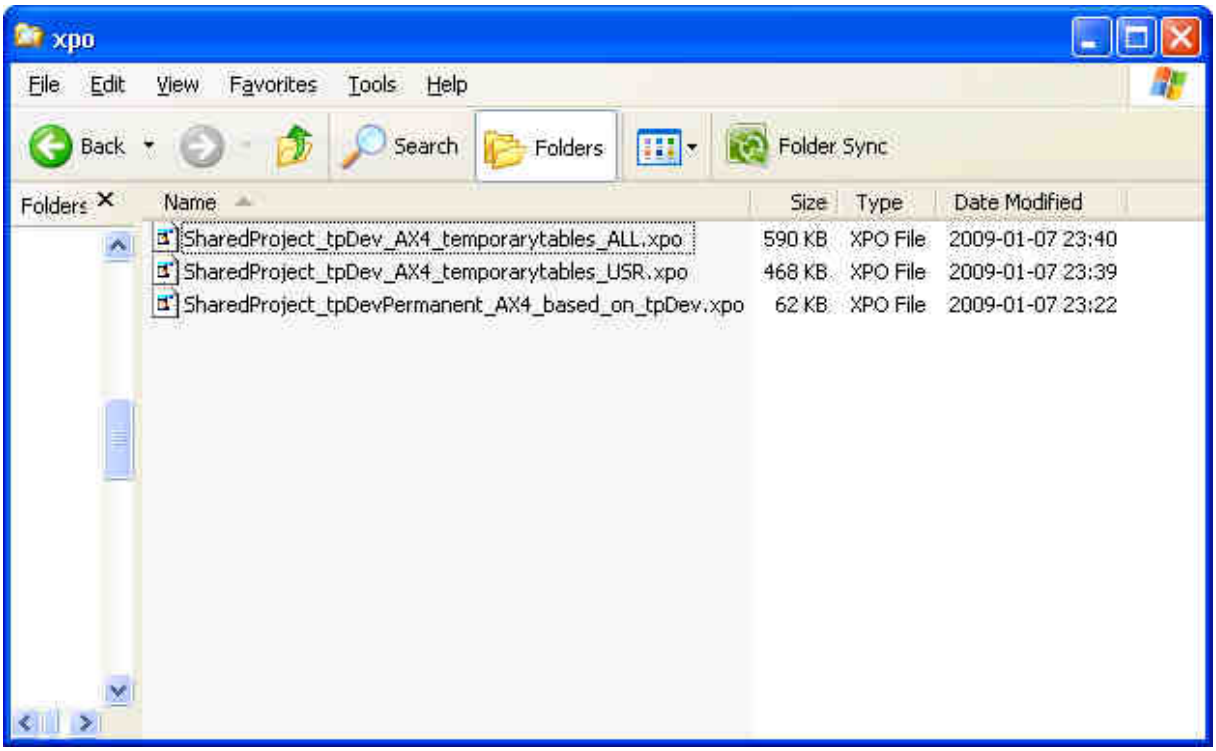


The WebService folder contains the following:


WebService
File Edit View Favorites Tools Help
Back Search Folders Folder Sync
Folders
Name S... Type Date Modified
example to build consumer from service.cmd.txt 1 KB Text Document 2008-12-04 19:41
Service1.asmx 1 KB ASP.NET Web Service 2008-11-27 12:32
Service1.asmx.cs 5 KB Visual C# Source file 2009-01-07 23:29
Web.config 2 KB XML Configuration File 2009-01-07 23:35

- An example file on how to compile and create a consumer for the web service. That consumer should be used in AX as a reference.
- Service1.* files are the methods exposed by the web service. They should be compiled and put on a web server.
- The web.config file contains information used in the Service1.* files. It is for user name and password for the business connector.

The xpo folder holds the xpo files used:


xpo
File Edit View Favorites Tools Help
Back Search Folders Folder Sync
Folders
Name Size Type Date Modified
SharedProject_tpDev_AX4_temporarytables_ALL.xpo 590 KB XPO File 2009-01-07 23:40
SharedProject_tpDev_AX4_temporarytables_USR.xpo 468 KB XPO File 2009-01-07 23:39
SharedProject_tpDevPermanent_AX4_based_on_tpDev.xpo 62 KB XPO File 2009-01-07 23:22

These files are for AX4 but there are only minor changes to AX2009. The first file is a complete xpo with all the elements modified. The second one is the same project, but with only the modifications from the working layer. The last xpo-file is an example with persistent data. It requires the first files to work.

5.6.1 Development Center AX2009

The prototype is based on Development Center AX2009 and some of the elements from this product have been modified.

5.6.2 References

You must add references to the business connector and the web service consumer dll that you have to create.

5.6.3 Business connector and Active Directory

The web service requires that the IIS can communicate with the Active Directory to authenticate the user. This may also be true for the business connector when used inside AX, depending on how credentials have been cached.

Chapter 6
Conclusion

In the beginning of the thesis a purpose was presented: to suggest improvements to the software configuration management routines across multiple applications of Microsoft Dynamics AX 2009. Some specified questions were also listed. I believe I have fulfilled the purpose and answered the questions. The challenge with the multiple environments for the companies in the study is the fact that none of them used a centralized repository, making it hard to track and trace what was happening in different environments. There were many places to work with tasks and elements, one in each environment. It was difficult to do status accounting.

What would be necessary to make software configuration management work across multiple Microsoft Dynamics AX 2009 applications? To answer this, I have found recommendations in the theories. What constitutes desirable features of a SCM system? What should be under configuration management? I have suggested a central repository, a Development Studio Server, where the tool can tie everything together, where all CIs can be tracked and traced. I have suggested what to treat as configuration items in the context of product development within Microsoft Dynamics AX 2009. General improvements to both issue tracking and current change management routines have been suggested.

SCM is an activity that should be part of the entire product lifecycle; it is an umbrella activity as Pressman says. A unified system across environments would be one large leap in the right direction, making it possible to get an overview of all bits and pieces of the solutions being developed.

With DevCenter built on Dynamics AX, no additional server technologies are needed. The prototype for DevCenter with a server setup meets requirements from most of the main functional areas defined chapter 2.5. It offers a tool with all data in one place – on the server, one shared repository and one place to manage all projects/products. Different client applications/environments can access the same data. It would even be possible to plug-in other areas as well (Dynamics NAV, .NET-development, etc.).

I have also suggested applying version control (revision control) to tasks and other documentation as well, thus creating traceability, auditability, and status accountability.

Putting elements, tasks, requirements, features, versions and products under configuration management is important, and to include other artifacts produced in the same unified tool, making it more streamlined to work; users can document the solution and bring everything together in one place.

Releases have been a time consuming issue. Improving auditability by managing releases and hotfixes properly makes is possible to know what is where when. It is a basic principle, and something that everyone agrees on takes a lot of time today. Don't look for what is where when. Make sure there are proper versions available that have a unique identifier and track where these versions have been installed. I do believe that DevCenter itself will make releasing solutions more efficient, but if possible I would suggest trying to the build of new releases further down the road.

Proper testing after something has changed is important. Track changes and their implications to make sure the outcome can be tested properly. Support for testing within the system would be a great improvement. Defects found early in the process save money.

No single tool however, will solve all the needs – it is not a silver bullet. To make SCM work across multiple applications also requires commitment from the organizations and a plan to follow.

Chapter 7
After study

The intermediate result of the thesis was presented to the stakeholders early in the process. After proving a server/client setup was possible to achieve without any third party solutions, work began with the new Development Studio.

The previous dependencies to existing components were removed and a stand-alone product to support SCM was created.

In late 2009 the first version of Development Studio Server for AX 2009 SP1 was released. It incorporated all of the findings and recommendations from the thesis. Figure 8 below is a picture from the user manual and describes the overall functionality and how Development Studio supports the requirements management, product management, development, testing and release management disciplines (Snaeland & van Veldhuizen, 2009).

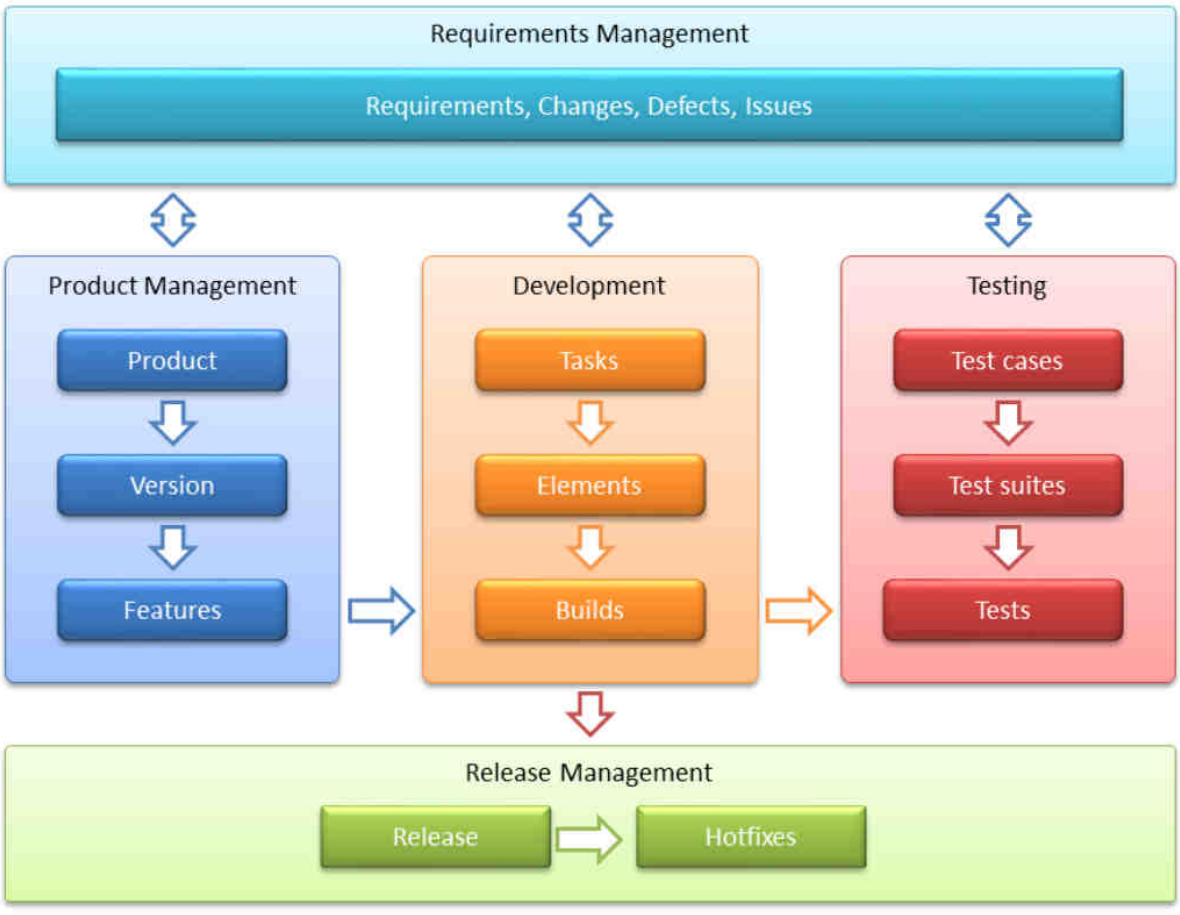


Figure 8 Development Studio features at a glance (Snaeland & van Veldhuizen, 2009)

Many improvements surrounding testing were introduced. Out of the box Development Studio supported tracking both manual and automatic tests. By defining test cases, test suites could be created for recording the results of test case execution. The key elements are described in figure 9 below. (Snaeland & van Veldhuizen, 2009)

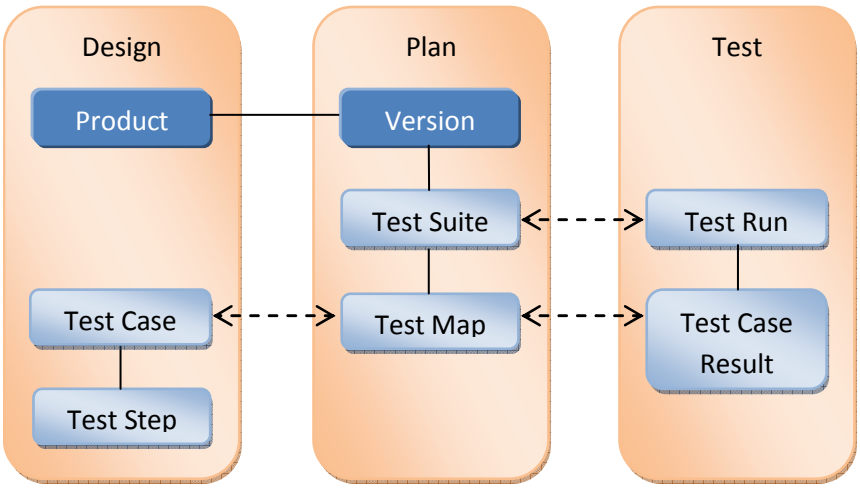


Figure 9 Key elements of testing features (Snaeland & van Veldhuizen, 2009)

As the components in the test system were reusable units, tests could be executed multiple times and reused between different versions ensuring that old functionality would not deteriorate when changes were introduced.

The release management that was identified as time consuming has become a lot more automated. Releases were added as a concept. A release was defined as a build that would be officially delivered to customers. In most cases, all version elements such as requirements, tests and tasks should have been closed, executed and all included documents be ready to ship at the time of release. With Development Studio it was easy to get a good overview. Released versions are treated like a new baseline, and no changes could be done to the release itself. However, a specific type of requirement labeled as hotfixes could be introduced and later be built and delivered to customers.

Documentation can easily be generated, both for releases but also for e.g. test protocols in Word etc. for printing. The server/client approach implemented in the end was using persistent data to support off-line functionality.

Development Studio was upgraded and is available also for Dynamics AX 2012 R3 and has been improved with even more features to support the development of products within Dynamics AX.

To-Increase has been selling Development Studio as a separate product after its release and also used it internally for several years. Annata is still using the product

for some of their applications in 2015, but has initiated a change in setup since technical possibilities and tools provided by Microsoft have changed for the better.

Chapter 8
References

Annata Sverige AB. (2008). *Om Annata*. Retrieved June 26, 2008, from Annata Sverige: http://www.annata.se/dync.aspx?id=200008

Avdic, A. (2005). *Forskningsmetodik AA – 3*. Retrieved March 13, 2005, from Forskningsmetodik AA – 3: http://www.oru.se/oru-upload/Institutioner/Ekonomi%20statistik%20och%20informatik/Dokument/Informatik/InformatikD/Forskningsmetodik/FoMet%207.pdf

Berlack, H. R. (1992). *Software Configuration Managment.* John Wiley & Sons, Inc.

Brown, W. J. (1999). *AntiPattern and Patterns in Software Configuration Management.* John Wiley & Sons, Inc.

Dart, S. (2007, January 11). Retrieved October 28, 2008, from Spectrum of Functionality in Configuration Management Systems: http://www.sei.cmu.edu/legacy/scm/tech_rep/TR11_90/TOC_TR11_90.html

Department of Defense - United States of America. (1988, July 15). *MIL-STD-480B, Configuration Control - Engerineering changes, deviations and waivers.* Retrieved September 22, 2008, from http://www.product-lifecycle-management.com/download/MIL-STD-480B.pdf

Department of Defense - United States of America. (1992, April 17). *MIL-STD-973, Configuration Management.* Retrieved September 22, 2008, from http://sepo.spawar.navy.mil/Mil-STD-973_CM.pdf

Gartner Inc. (2015, May 14). *SCM (software configuration management)*. Retrieved May 14, 2015, from Technology Research | Gartner Inc.: http://www.gartner.com/it-glossary/scm-software-configuration-management

Leon, A. (2000). *A Guide To Software Configuration Management.* Artech House.

Microsoft Corporation. (2002, July 11). *Microsoft Acquires Navision*. Retrieved May 14, 2015, from Microsoft: http://news.microsoft.com/2002/07/11/microsoft-acquires-navision/

Microsoft Corporation. (2008). *ERP software: Microsoft Dynamics AX product information*. Retrieved September 17, 2008a, from http://www.microsoft.com/dynamics/ax/product/default.mspx

Microsoft Corporation. (2015). *Microsoft Dynamics AX: ERP Business Software*. Retrieved May 14, 2015, from Microsoft Corporation: http://www.microsoft.com/en-us/dynamics/erp-ax-overview.aspx

Moore, J. W. (1998). *Software Engineering Standards - a User's Road Map.* IEEE Computer Society.

Nationalencyklopedin. (2008). *Nationalencyklopedin*. Retrieved March 13, 2005, from http://www.ne.se/jsp/search/article.jsp?i_art_id=167073

Pressman, R. S. (2005). *Software Engineering - A Practitioner's approach* (6th ed.). New York: McGraw-Hill.

Royce, W. (1998). *Software Project Management, a Unified Framework.* Addison Wesley Longman, Inc.

Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering 14(2)*, 131-164.

Snaeland, P., & van Veldhuizen, J. (2009, September 3). User manual - Development Studio.

To-Increase. (2008). *To-Increase at a glance*. Retrieved June 26, 2008, from http://www.to-increase.com/Our_Company/47/To-Increase_at_a_Glance.html

På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Martin Cederbom